

On Replacing Cryptopuzzles with Useful Computation in Blockchain Proof-of-Work Protocols

ANDREA MERLINA, THIAGO GARRETT, and ROMAN VITENBERG, University of Oslo, Norway

Proof-of-Work (PoW) blockchains have emerged as a robust and effective consensus mechanism in open environments like the Internet, leading to widespread deployment with numerous cryptocurrency platforms and substantial investments. However, the current PoW implementation primarily focuses on validating the discovery of a winning nonce. Exploring the notion of replacing cryptographic puzzles with useful computing tasks becomes compelling, given the substantial computational capacity of blockchain networks and the global pursuit of a more sustainable IT infrastructure. In this study, we conduct a comprehensive analysis of the prerequisites for alternative classes of tasks, examining proposed designs from existing literature in light of these requirements. We distill pertinent techniques and address gaps in the current state-of-the-art, providing valuable insights into the evolution of consensus mechanisms beyond traditional PoW.

Additional Key Words and Phrases: Blockchain, Useful Computation, Proof-of-Work

1 INTRODUCTION

Since Bitcoin’s [88] inception in 2008, the system has attracted numerous stakeholders, growing in size and reaching a market cap of over 400 billion dollars [27]. A central element in Bitcoin’s design is the Proof-of-Work (PoW) mechanism for block proposals. PoW secures the chain, limits Denial-of-Service attacks, and improves the consensus algorithm scalability with respect to the participants. However, as it has been widely publicized and criticized, the energy consumption of PoW in Bitcoin is comparable to medium-sized countries. Indeed, at the time of writing, Bitcoin consumes as much annual energy as Belgium [19]. While a number of alternative and less energy-consuming mechanisms have been proposed [90], PoW remains a dominant mechanism in permissionless blockchains.

What makes the PoW mechanism in Bitcoin and other blockchain systems particularly wasteful is that it is based on solving cryptopuzzles. A solution to a cryptopuzzle does not serve any purpose or has any value outside of the blockchain ecosystem; one could argue that the hard-sought-after solutions to cryptopuzzle do not benefit humanity whatsoever. At the same time, in other contexts, a vast number of computational tasks need to be solved, whether in operational research, computational biology, Machine Learning, or other optimization problems.

The alluring dream of replacing useless cryptopuzzles with useful work has been tantalizing researchers for a while, motivating a large body of work in the area. However, despite numerous attempts and important academic contributions, no such system has been deployed yet. Where does state-of-the-art fall short, and can we still hope for such a replacement? Our paper aims to address this question systematically.

Cryptopuzzles are exploited by the block proposal mechanism in Bitcoin and other systems where PoW grants the miner the right to propose a block. The block proposal mechanism plays a central role in ensuring correct system operation expressed through the well-known holistic requirements such as Common prefix, Chain quality, and Chain growth [43]. We start by identifying the block proposal properties (BPPs) that lead to fulfilling the holistic requirements. Subsequently, we identify those properties of cryptopuzzles that are exploited by the block proposal mechanism toward satisfying the formulated BPPs. We observe that neither BPPs nor cryptopuzzle properties exploited in Bitcoin have been covered in the literature in a comprehensive or standardized way,

yet each individual proposed scheme has been presenting a partial list using different specifications. To the best of our knowledge, this is the first work to present a comprehensive list of BPPs and exploited cryptopuzzle properties (including those that have never been mentioned in the literature) and analyze them in detail. In order to create a uniform framework, we isolate the PoW component in Bitcoin, present the component’s interface, and use the interface to express the properties. For each property, we explain why it is needed for BPPs, how it is satisfied by cryptopuzzles, and how challenging it would be to satisfy it when using other classes of tasks. Although we do not prove our list of properties to be complete, it is more comprehensive than the previously known set of properties and can be used to assess the feasibility of alternative classes of tasks and novel PoW-based designs.

This property analysis allows us to consider the transition to a PoW system that is based on tasks other than cryptopuzzles. We discuss how the blockchain architecture and the interface of the PoW component need to be adapted to this end. We present two alternative definitions of “usefulness” in the context of PoW tasks and the need for a task supply model. Then, we systematically analyze the challenges posed by replacing cryptopuzzles with general classes of useful tasks. A number of those challenges are due to the introduction of usefulness, whereas the other challenges are encountered because some of the cryptopuzzle properties are difficult to replicate with other classes of tasks.

In light of this challenge analysis, we survey a range of proposals in the literature. To this end, we identify three popular classes of tasks (based on the problems of k -Orthogonal Vectors, Traveling Salesman, and Deep Learning, respectively), which have been touted as candidates for replacing cryptopuzzles in past proposals. After considering how well each class suits each PoW-induced requirement, we provide systematic coverage of individual proposals.

Our analysis reveals that usefulness is in direct conflict with many important properties. Moreover, no alternative task or system design that can satisfy all properties and completely replace cryptopuzzles has been proposed.

In summary, the contributions of this paper are as follows:

- (1) We systematically define, categorize, and discuss important properties that characterize blockchain tasks such as cryptopuzzles. The resulting framework of properties and the related discussion are of interest to both researchers and developers as a blueprint for the analysis and design of new systems.
- (2) We consider the effect of introducing “usefulness” and of transitioning to task classes other than cryptopuzzles and analyze the implications for the blockchain architecture and for the identified properties.
- (3) We categorize classes of tasks considered in the literature as candidates for replacing cryptopuzzles and provide an in-depth analysis of the state-of-the-art for three of these classes.
- (4) We review and classify several proposed designs in the literature with respect to the identified properties, showing in practice how the proposed property framework can be used as a guide for both blockchain developers and researchers to assess the effectiveness of proposed designs.
- (5) Finally, we identify research gaps in the state-of-the-art and outline a number of future research directions.

The paper is structured as follows. In Section 2, we provide the necessary background on computational tasks, Bitcoin, and PoW. In Section 3, we define and analyze the properties of classes of tasks that are necessary to support identified PoW requirements. After presenting the implications of usefulness in the context of PoW in Section 4, we describe three popular classes of tasks in Section 5 and consider how well they satisfy the defined properties. This is followed by an

analysis of several proposed designs in Section 6. We present a list of research gaps in Section 7, cover related work in Section 8, and conclude in Section 9.

2 BACKGROUND

Section 2.1 presents definitions for computational tasks. Next, Section 2.2 presents a high-level overview of Bitcoin (the first and perhaps the most representative PoW-based blockchain system), which is used as a base model throughout this work. Section 2.3 then describes PoW and cryptopuzzles in Bitcoin. Finally, we present the formal requirements of the consensus problem solved by Bitcoin in Section 2.4.

2.1 Computational Tasks

A *task* is a problem requiring non-trivial computation to produce a solution. In this work, such non-triviality is often referred to as *hardness*. A *class of tasks* is the type of problem being solved. Examples of classes of tasks include linear algebra operations, satisfiability (SAT), and the Traveling Salesman Problem [30] (TSP). Thus a task consists of a specific instance of a class of tasks, e.g. a specific weighted graph configuration in the case of TSP. Solving a task implies executing a solution algorithm to find a solution to the task instance. Algorithms for solving different classes of tasks have different computational complexity, as extensively surveyed in [44].

In the context of this work, solving a task produces a Proof of Computation (PoC) that serves two purposes: (a) a solution to the task can be efficiently derived from the PoC, and (b) the PoC serves as a proof allowing an external party which did not take part in solving the task to verify that the task was solved correctly. We call the process in (b) task verification.

Next, we briefly provide a formal definition of three relevant classes of tasks.

k-Orthogonal Vectors. Linear algebra operations, among which the calculation of orthogonal vectors, find applications in many disciplines, such as traffic flow and electric circuits. The *k-Orthogonal Vector (k-OV)* problem is defined as follows [8].

Definition 1 (*k-Orthogonal Vectors*). The *k-OV* problem on vector of dimension d is to determine, given k sets $(U_1, \dots, U_k)$ of n vectors in the form $\{0,1\}^{d(n)}$, which $u^s \in U_s$ for each $s \in [k]$ such that over $\mathbb{Z}$:

$$\sum_{l \in [d(n)]} u_l^1 \cdots u_l^k = 0 \quad (1)$$

The solution to a *k-OV* task is, therefore, a list of OVs. No reduction is known from *OV* to *CNF-SAT*, and it is commonly conjectured that any algorithm requires $n^{k-o(1)}$ [123]. For the restricted case of $k = 2$, the complexity becomes $n^{2-o(1)}$. Obtaining a truly sub-quadratic algorithm for *OV* has been elusive, and [122] showed that the Strong Exponential Time Hypothesis (SETH) [61] implies the nonexistence of such an algorithm.

Traveling Salesman Problem. Solutions to TSP tasks find applications in areas such as logistics by planning the shortest route among interest points, minimizing the cost of circuit board printing, as well as planning the movements of bulky astronomy telescopes, which are slow to re-target. TSP is a classic example of an NP-complete problem for which, due to its hardness, there are several longstanding instances whose optimal solutions are not known [56, 92]. It is defined as follows [47].

Definition 2 (*Traveling Salesman Problem*). Given a set $C = \{c_1, \dots, c_n\}$ of *cities*, and a distance function $d : C \times C \rightarrow \mathbb{N}$, the solution to the Traveling Salesman Problem is a cycle that visits every city once, also known as a Hamiltonian cycle, resulting from a permutation of the cities $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$, that minimizes the cost of traveling expressed as:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \quad (2)$$

The Traveling Salesman Problem can be formulated as a decision problem by introducing a threshold t_d so that a solution is valid if and only if the cost of the valid permutation d_{π} is such that $d_{\pi} \leq t_d$.

The fastest exact solver of TSP is considered to be Concorde [29]. On the other hand, several heuristic algorithms such as LK [78] and LKH [53] offer quicker and generally good quality solutions [54]. A standard benchmark for testing those algorithms is the instance library TSPLIB [101]. TSPLIB introduces a format [91] as well, by which cities are identified in two-dimensional coordinates, and every distance pair is calculated with the Pythagorean theorem.

Deep Learning. Machine Learning (ML) investigates methods that leverage data to improve the performance of a set of tasks. While the first ML methods date back to 1960, improved data access and hardware performance paved the way for more complex methods such as Deep Learning (DL), which is the subject of extensive research due to its successful track in tackling advanced tasks spanning from computer vision to machine translation.

A Neural Network (NN) is a mathematical model of functional transformations often represented as a directed graph where edges have tunable weights and vertices, typically referred to as neurons, perform linear and non-linear transformations. NNs have been successfully employed in various applications, including supervised learning, which is described as follows. Given a training set of N example input-output pairs $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$, where each pair is generated by an unknown function $y = f(x)$, the task of supervised learning is to discover a function h_w that approximate the true function f [105]. Deep Learning (DL) encompasses a broad family of techniques in which NN layers are arranged in sufficiently significant numbers, hence the name Deep Neural Network (DNN). DL tasks are defined as follows [12].

Definition 3 (Deep Learning Task). A Deep Learning Task consists of the process of learning a function $h_w : X \rightarrow Y$, where h_w is parameterized by a vector $w \in W$ representing the tunable weights. The performance of h_w on an example (x_i, y_i) is measured by an error function $E(w)$ calculated across all input-output pairs. Therefore the goal is to minimize the following:

$$E(w) = \frac{1}{2} \sum_{n=1}^N \| h_w(x_n, w) - f(x_n) \|^2 \quad (3)$$

The error $E(w)$ is a smooth continuous function of w that assumes the smallest value when the gradient vanishes, i.e. $\nabla E(w) = 0$. Stochastic Gradient Descent (SGD) techniques make an update to the weight vector based on one data point at a time so that:

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \quad (4)$$

where the parameter $\eta > 0$ is known as the *learning rate*. After each update, the gradient in Equation 4 is re-evaluated for the new weight vector and the process repeats, in what is commonly referred to as epochs. In the rest of the work we refer to NN error, performance and accuracy interchangeably, to express how close h_w approximates the true function f .

2.2 Bitcoin

In Bitcoin, multiple nodes exchange messages over a partially synchronous and sparse peer-to-peer (P2P) network. The system is *permissionless* so that nodes are free to join or leave, and the

complete set of participating nodes is unknown. Messages are broadcast following a gossip-based protocol [34]. An asymmetric cryptography scheme is assumed by which all messages are signed to ensure authenticity and integrity. There are two types of messages: transactions and blocks. A transaction transfers the ownership of a resource from a sender to a recipient, both represented by public keys. A block is a data structure that contains an ordered set of valid transactions in addition to metadata. A valid transaction transfers resources that were not transferred before – an attempt to transfer the same resource more than once is called *double-spending* and is considered to be an attack on the system. The nodes creating transactions to be included in blocks are called *clients*. The system’s goal is to maintain an ordered, durable, and replicated history of blocks – a blockchain, described next.

A blockchain is an ordered and immutable sequence of blocks, and it is a type of distributed ledger. The system maintaining a blockchain, such as Bitcoin, is called a *blockchain system*. The position of each block in the blockchain is called *block height*. Each block contains in its metadata the hash value of the previous block – Bitcoin, for instance, uses the SHA-256 hash function. This cryptographic pointer to the previous block ensures that any modification in a block will modify the hash value of all following blocks in the chain. Thus in this work, we only consider a linear chain of blocks instead of other alternative structures such as directed acyclic graphs [117]. Blocks may be created by any node participating in the system. But for a block to be valid, its metadata must also include proof that the corresponding node had the right to create the block. Nodes compete for the right to create and append the next block of the blockchain by solving a cryptopuzzle, following a PoW consensus protocol – the cryptopuzzle and how the consensus protocol handles conflicting block proposals are described later in this section. Nodes participating in the block proposal competition are called *miners*. A miner that *appends* a block (i.e. creates a block that eventually becomes part of the blockchain) is rewarded by financial means: the miner inserts a special self-rewarding transaction called *coinbase transaction* in the proposed block. Nodes verifying the validity of blocks are called *validators* – all miners are also validators.

A certain fraction of nodes can behave arbitrarily, i.e. the presence of Byzantine faults is assumed [72]. The PoW protocol requires nodes to spend computational power to solve cryptopuzzles. In this context, an essential security assumption states that the honest nodes constitute more than half of the total computational power of the system. The immutability of the blockchain relies on this assumption. The specific cryptopuzzle that needs to be solved for each block depends on the hash value of the block, as described in Section 2.3. Thus to modify a block (e.g. to double-spend a resource), the PoW for the modified block and all following blocks must be recomputed faster than the rate at which the rest of the system creates new blocks. This is shown to be unfeasible under the security assumption above [88]. To incentivize nodes to join the system and behave honestly (i.e. following the standard protocol), a miner receives two rewards for appending a block to the blockchain: block reward and transaction fees. Block reward consists of newly created resources given to a miner that created an appended block. At the same time, a transaction fee is a payment by the creator of a transaction to the miner that includes the transaction in an appended block. Moreover, transaction fees are a protection mechanism against Denial-of-Service (DoS) attacks [4] since creating many transactions to flood the network and prevent nodes from performing tasks other than validating transactions would be too costly due to the fees.

The Bitcoin model described in this section may be divided into five layers, as shown in Figure 1: Hardware, Communication, Storage, Consensus, and Application. The Hardware layer consists of the physical devices on which software implementing the blockchain system runs. In Bitcoin, it is common to employ ASICs and purpose-built hardware for solving cryptopuzzles, in addition to general-purpose hardware for other uses, such as storing the blockchain and communicating with other nodes. The Communication layer is responsible for connecting nodes in an overlay

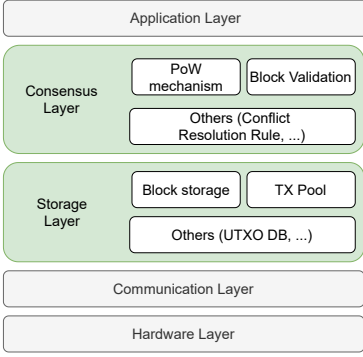


Fig. 1. Bitcoin layers.

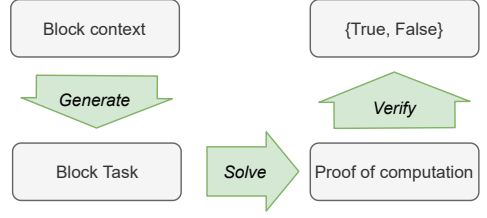


Fig. 2. Base interface. Boxes represent the inputs and outputs of the interface primitives, shown as arrows.

P2P network, on top of which nodes broadcast messages following a gossip-based broadcast protocol. The Storage layer deals with storing the blockchain and other data necessary for validating transactions and creating blocks. The Consensus layer guarantees that all honest nodes eventually agree on a single valid blockchain. Finally, the Application layer comprises the services that can be built using the blockchain system. Bitcoin, for example, supports payment applications, such as Lightning [98]. For more information on these layers, we refer the reader to [36, 89, 124]. This work focuses on the Storage and Consensus layers, further described below.

In Bitcoin, many components are maintained by the Storage layer. In this work, we focus on two: Block storage and Transaction (TX) Pool. The *Block storage* component stores valid blocks that were received or created by the node. This includes the blockchain and potentially conflicting blocks currently not part of the blockchain, as per the consensus protocol – the *orphaned blocks*. Note that a node may store only a subset of the blockchain. The *TX Pool* stores all pending transactions, i.e. transactions not included in any proposed block yet. Each transaction contains a list of inputs and a list of outputs. Inputs are the outputs from other past transactions, which the current transaction is spending (i.e., transferring the corresponding resources). Outputs represent the recipients of the transferred resources (which come from the inputs). Examples of other components in the Storage layer include the Unspent TX Output Database (UTXO DB). This component stores unspent outputs i.e. outputs that were not used as inputs in any transaction already included in the blockchain. This component is essential for efficiently validating transactions: a transaction is only valid if its inputs are in the UTXO DB.

In this work, we focus on two components maintained by the Consensus layer: the PoW mechanism and Block Validation. The *PoW mechanism* is responsible for producing a PoW (by spending computational resources) for a miner to be able to propose a valid block. Section 2.3 describes this mechanism in more detail. A block proposed by a miner is broadcast to all other nodes, which validate the block through the *Block Validation* component. For a block to be valid, it must contain: (i) the hash of the previous block in the chain; (ii) a set of valid transactions; and (iii) a valid PoW, i.e. the solution for the correct instance of a cryptopuzzle. Upon receiving a valid block, an honest node adds the block to its local blockchain replica. Since these two components are executed simultaneously by multiple nodes in the system, several blocks pointing to the same previous block may be proposed. This creates a *fork* in the blockchain, resulting in multiple concurrent *branches*. Different miners may continue proposing blocks extending different branches. Therefore, in the case of a fork, there are many different versions of the blockchain. Reaching consensus in this scenario means that all honest nodes agree on which branch is the main chain, discarding the

blocks in the other branches (which become *orphaned blocks*). This agreement is achieved through the Conflict Resolution component. In Bitcoin, this component enforces the *longest chain* rule, which considers the branch with the largest number of blocks as the main chain. The rationale is that one of the concurrent branches will eventually become longer than the others due to the time variability of the PoW mechanism, described later in this work. In this scenario, increasingly more miners extend the same branch and *converge* to the same chain. The longer such branch becomes, the lower the probability that another branch becomes longer. Therefore, the consensus in Bitcoin is probabilistic rather than deterministic [115]. Although in this work we do not focus on the Conflict Resolution component, it is included, together with PoW mechanism and Block Validation, in the so-called Bitcoin *backbone*, i.e. how blocks are proposed, validated, and accepted by nodes in the system to build the blockchain. The consensus problem solved by this backbone is further described next in Section 2.4.

2.3 PoW and Cryptopuzzles in Bitcoin

In a Proof-of-Work (PoW) protocol [64], an entity (the prover) demonstrates to another entity (the verifier) that a certain amount of computational work was performed. PoW is a well-established mechanism to prevent Sybil and DoS attacks since PoW makes it computationally expensive for an attacker to create many identities and/or requests. In Bitcoin, PoW is also crucial for regulating the rate at which blocks are proposed by miners, in addition to DoS and Sybil protection. It takes, on average, 10 minutes for at least one miner in Bitcoin to propose a block containing a valid PoW. The importance of controlling the block proposal rate is further elaborated in Section 3.

The PoW protocol in Bitcoin consists of solving a cryptopuzzle based on Hashcash [5] and embedding the solution in the proposed block. Bitcoin's PoW finds a value – the *nonce* – such that the result of a hash function applied to the concatenation of the nonce with a given input data is smaller than a pre-defined target value – the *difficulty*. Therefore, solving the cryptopuzzle for a given input data involves sequentially computing a hash function, varying only the nonce. The proof consists of the input data and the nonce, which can be verified efficiently by computing the hash function only once and comparing the resulting value with the expected difficulty. In the context of this work, Bitcoin's cryptopuzzle is a class of tasks. Historically, the use of hash functions to prevent content spamming was first proposed by Dwork and Naor in 1993 [38].

The input data for a cryptopuzzle includes two values: (i) the hash of the root of the Merkle tree containing the set of transactions to be included in the block being created; and (ii) the hash of the previous block in the chain. We call this input data *block context*. Merkle tree is the data structure used in Bitcoin to organize the transactions included in a block, which are taken from the TX Pool. To be able to propose a newly created block, a miner tries to solve the cryptopuzzle task. The proposed block thus contains the block context, the nonce, and the Merkle tree. The process of solving a cryptopuzzle to be able to propose a block is often called *mining*.

Cryptopuzzles have some unique features when compared to other classes of tasks. Given the nature of cryptographic hash functions, the best available algorithm to solve cryptopuzzles is to try, in a brute-force fashion, many possible values for the nonce. The probability of any nonce value to be the solution is the same, and the solving process is *memoryless*, i.e. the probability of any nonce value to be the solution does not change, regardless of how many other values have already been evaluated. Typically, miners try so many nonces that Bernoulli trials, a discrete probability process, can be well approximated by a Poisson process, which is instead continuous. At the network level, the resulting time between block proposals is exponentially distributed, as formally shown in [13]. In Bitcoin, the nonce is a 32-bits value. In case a miner evaluates all 2^{32} possible values for the nonce without solving the cryptopuzzle, the miner may then modify the block context by, for example, changing the order of some transactions in the Merkle tree of the block being mined, resulting in a

new cryptopuzzle instance to be solved. In that sense, a cryptopuzzle is considered computationally intensive yet solvable. We further discuss many properties satisfied by cryptopuzzles in the context of this work next in Section 3.

2.4 The Problem Solved by the Bitcoin Backbone

In this work, we identify several properties related to cryptopuzzles in Bitcoin and map such properties to computational tasks in general. As a foundation for these properties, formal definitions of the requirements of the Bitcoin backbone are necessary. At a high level, Bitcoin solves the consensus problem in an open and permissionless environment. The formalization of this problem is not a trivial task by any means; it has been explored in several comprehensive and representative works [43, 68, 95, 110]. At the level of the ledger abstraction, the two main properties are (a) persistence: if a transaction gets added to the public ledger, it never gets removed and (b) liveness: if an honest node wants to add a transaction to the ledger, the transaction should eventually be included in it [43, 95]. However, the PoW protocol does not consider individual transactions or metadata, blocks are secured as atomic entities. Therefore, we do not consider the ledger abstraction in this work; instead, we focus on the underlying *Bitcoin backbone abstraction* and its properties, which are more relevant to the problem of replacing cryptopuzzles with other tasks.

The Bitcoin backbone problem includes the following requirements [95]:

- Common Prefix:** at any point, the chains of two honest nodes can differ only in the last T blocks with overwhelming probability, which depends on the parameter T ;
- Future Self-Consistency:** at any two points r, s during the execution, the chains of any honest node at r and s differ only within the last T blocks with overwhelming probability, which depends on the parameter T ;
- Chain Quality:** for any T consecutive blocks in any chain held by some honest node, the fraction of blocks that were contributed by honest miners is at least μ with overwhelming probability, which depends of the parameters T and μ ;
- Chain Growth:** at any point in the execution, the chain of any honest node grows at a rate that does not fall below the “minimum rate” with overwhelming probability; the parameters of minimum rate and overwhelming probability need to be further specified, but this specification is not germane to the problem of replacing cryptopuzzles.

Intuitively, the properties of Common Prefix, Future Self-Consistency, and Chain Quality contribute to the safety and security of the Bitcoin system: they stipulate that bad scenarios never occur and that important invariants are never violated, even under attacks. On the other hand, chain growth is vital for a stable progress rate in Bitcoin.

The authors of [43] and [95] prove that the Bitcoin implementation solves the backbone problem under a number of specific assumptions about the environment in which the system operates and about the computational capabilities of the nodes. Since replacing cryptopuzzles with useful computation does not affect those assumptions and other aspects of the operational environment, discussing these assumptions is beyond the scope of our work.

The scope of formalization in the Bitcoin backbone problem is somewhat narrower than the entire Bitcoin system, as fairly observed by the authors of [43]. In particular, the backbone problem does not consider node incentives due to the assumption that all nodes are statically divided into honest and malicious. In practice, the problem that Bitcoin solves has an additional requirement stipulating that the nodes are incentivized to participate in the system and that two computationally weaker miners do not gain higher rewards from combining their forces and acting as a single computationally stronger miner. This requirement prevents undesirable centralization in Bitcoin and boosts security in practice.

3 WHY CRYPTOPUZZLES WORK FOR POW

Despite the conceptual simplicity of PoW, cryptopuzzles and their use in the PoW mechanism satisfy surprisingly many properties. The goal of this section is to gain insight into these properties and show how they support the Bitcoin backbone protocol and other Bitcoin requirements listed in Section 2.4. Later, in Section 4, we extend these properties to cover general computational tasks. We use the term *Block Task* (BT) to refer to general computational tasks used to secure blocks; cryptopuzzles are, for instance, one example of BT. Our methodology is as follows: first, we define a layer of properties of Block Proposals (BP) and show how they support Bitcoin requirements. Then, we define a layer of Block Task Properties (BTP) and show how they support Block Proposal Properties (BPP). In order to define the BTPs, we first formalize an interface for BTs in the context of PoW and associate the properties with individual primitives in the interface.

While BTPs and the Bitcoin requirements listed in Section 2.4 are presented formally, the properties of BP are stated at a more intuitive level. The goal of our paper is to motivate individual BTPs, connect them to Bitcoin requirements, and analyze how cryptopuzzles and other classes of tasks may satisfy these task properties. In the context of this work, BPPs mostly play a role in connecting task properties to Bitcoin requirements; it is not our goal to provide a specification for BPP. To maintain a wide applicability, we employ generic terms such as “too little” or “too much” time. Quantification of these terms depends on the values of parameters in the backbone protocol, such as the Chain Growth rate. A precise definition would require specific assumptions on the system and models, which is outside the scope of this paper. For the sake of readability, we group all properties into three categories. *Safety and security* properties support the requirements of Common Prefix, Future Self-Consistency, and Chain Quality. *Liveness* properties are used to guarantee Chain Growth while *Decentralization* properties prevent the risk of centralization as described in Section 2.4. Table 1 gives an overview of the properties and their relation to the interface. Next, we start by defining the interface.

3.1 Block Task interface

The life cycle of BTs in Bitcoin-based blockchains is as follows. A miner partially forms a new block, thereby creating a block context (see Section 2.3). Using the block context, the miner *generates* an associated BT and attempts to *solve* such BT. If successful, the miner inserts the solution into the previously formed block, creates a BP, and broadcasts the BP to other miners. Upon receiving the BP from the network, the validators also *generate* a BT using the block context and *verify* the validity of the solution included in the BP w.r.t. the BT.

All operations in the cycle are local, apart from broadcasting and receiving the BP from the network. This cycle can be interrupted at any moment, e.g., by the arrival of new blocks from the network, which may result in updating the block context. In such scenarios, the cycle is started anew. In the following, we focus on the generation, solution, and verification of BTs, leaving aside elements relevant to BPs but not for BTs, such as the selection of transactions to be included in the block or their validation. Table 2 and Figure 2 show the inputs and outputs of the three operations in the interface.

3.2 Safety and Security Properties

The high-level goal of safety and security properties is to prevent incorrect system states. In the context of blockchain, a system is secure if blocks are correctly appended to the chain (refer to the Common Prefix, Future Self-Consistency and Chain Quality properties). One particularly harmful scenario for safety and security is the proliferation of chain forks and increased branching (which in the following are used interchangeably) because forks and branching invalidate the Bitcoin

Table 1. Overview of Block Proposal and Block Task Properties

Classes of Properties	Block Proposal Properties (BPP)	Block Task Properties (BTP)	Interface
Safety & Security	Limited BP rate	(a) BT hardness (b) Context sensitivity (c) Non-amortizability ¹	<i>Generate</i> <i>Generate</i> <i>Solve</i>
	Non-zero BP variability	Non-zero variability across BTs	<i>Solve</i>
	Adjustable lower threshold for difficulty	Adjustable lower threshold	<i>Generate</i>
	Block switchability	(a) No reduction in solv. (b) Neg. BT gen. time (c) No incr. in sol. time	<i>Generate</i> <i>Generate</i> <i>Solve</i>
	BP verification soundness	BT verification soundness	<i>Verify</i>
Liveness	BP verification efficiency	(a) BT generation efficiency ² (b) BT verification efficiency	<i>Generate</i> <i>Verify</i>
	BP timeliness	(a) BT generation efficiency ² (b) BT solvability (c) BT tractability	<i>Generate</i> <i>Generate</i> <i>Solve</i>
	Adjustable upper threshold for difficulty	Adjustable upper threshold	<i>Generate</i>
	BP verification completeness	BT verification completeness	<i>Verify</i>
Decentralization	BP fairness	(a) No sup. dep. on res. (b) Non-amortizability ¹	<i>Solve</i> <i>Solve</i>

¹ Same BTP for *Limited BP rate* and *Proposal fairness*² Same BTP for *BP verification efficiency* and *BP timeliness*

Table 2. Block Task Interface

	Generate	Solve	Verify
Input	Block Context	Block Task	Block Task, Proof of Computation
Output	Block Task	Proof of Computation	Boolean
Descr.	Generates a Block Task such as a cryptopuzzle instance based on the block context. Generation is performed independently by every miner and validator.	Search for a solution that satisfies the validity requirements.	Verifies that the proof of computation (previously produced by Solve) is indeed a valid solution for the Block Task.

backbone properties described in Section 2.4. With many forks, the probability of miners receiving blocks in different order increases. Therefore, the divergence of the chains of two honest miners probabilistically increases and the probability to violate Common Prefix by exceeding the last T blocks is higher. Similarly, high branching increases the risk of violating Future Self-Consistency. Another effect of forks is reduced mining utilization [39] i.e. reduced amount of computing power securing the chain. Lower mining utilization creates a risk of scenarios in which several consecutive blocks are appended by malicious nodes, invalidating the Chain Quality property. Hence, increased branching harms the security and safety of blockchain.

In the following, we intuitively describe five important BPPs for security, namely *Limited Block Proposal rate*, *Non-zero Block Proposal variability*, *Adjustable lower threshold for difficulty*, *Block switchability* and *Block Proposal verification soundness*.

Block Proposal Property 1 (Limited Block Proposal Rate). *The probability that a valid block proposal takes “too little” time to form is “sufficiently” small.*

A fundamental requirement for safety is that blocks cannot be proposed or appended to the local copies of the ledger at too high of a rate. With a rate too high, the probability of branching increases, thereby affecting security. In general, the proposal rate that maintains branching at an acceptable level depends on other parameters such as block size and information propagation mechanisms, as extensively studied in other works [45]. Besides, PoW-based blockchains may tolerate a small number of such fast proposals, but the probability of their creation needs to be low. We identified three BTPs relevant for BPP 1.

BTP 1a (BT hardness): Given an arbitrary implementation I of the *Solve* function, the probability that *Generate* returns a “simple” BT, that is a BT on which an invocation of I takes too little time to run is low. It is obvious that the existence of many simple BTs invalidates BPP 1.

BTP 1b (Context sensitivity): The BT produced by *Generate* depends on the input block context. Specifically, given two different block contexts, the probability to *Generate* the same BT for each is very low. The rationale for BTP 1b is to prevent unwarranted proposals that are, for example, reusing already solved instances or stealing the solution of another miner. Since it is possible to develop a dictionary of past BT solutions, BTP 1b strictly limits the possibility that miners submit valid proposals without performing any work. This would increase the rate of valid proposals, invalidating BPP 1 regardless of the hardness of the reused BT. In particular, this property means that the space of BTs produced by *Generate* is ample. Interestingly, BTP 1b does not protect from the case in which the miner can gain knowledge about the solution from outside the system. For example, there could be an external database of all known instances and solutions. Thus, the space of BTs produced by *Generate* needs to be much larger than the space of known solved instances. In Bitcoin, BTP 1b holds due to the enormous space of values that the block context might assume: the probability of generating the same BT is thus overwhelmingly low.

BTP 1c (Non-amortizability): *Solve* cannot be amortized across a set of different BTs so that the process of solving BT A provides no speed-up for the subsequent process of solving BT B . If the solution process can be amortized, the time to create a valid BP may be reduced in practice, especially for a miner that has been solving BTs for a long time. For example, computing a given Fibonacci number can be efficiently sped up by the knowledge of lower-value Fibonacci numbers. In another example, the solution to the Orthogonal Vector problem is the number of vectors in a given set whose dot product is equal to zero. Suppose multiple tasks include input sets with equal vectors. In that case, the computation across tasks can be amortized: knowing the solution for two sets of vectors removes the need to compute it again for the other sets. Similarly, if some specific configurations are known to repeat, miners may build dictionaries of (intermediate) solutions to amortize the block proposal process. For instance, in Machine Learning, it is common practice to reuse semi-trained models to shorten the training time. The amortization issue applies to many types of tasks, and it is exacerbated in systems where external clients supply arbitrarily tasks. In Bitcoin, on the other hand, BTP 1c is guaranteed by the nature of cryptopuzzles, for which the solution search is memoryless, and therefore no private state can speed up the search.

Block Proposal Property 2 (Non-zero Block Proposal variability). *Given an arbitrary chain, the times at which different miners produce competing BPs for the next block exhibit non-zero variability.*

In PoW-based blockchains, BP formation is a stochastic process. Bitcoin’s block proposal, for example, is modeled as a Poisson process [34] whose time between blocks follows an exponential distribution with a 10 minutes mean value, as shown in Figure 3. The variability of BP limits the number of conflicting blocks in the network at any time, which reduces the number of forks. Ideally, no two proposal solutions should be found at approximately the same time. However, the amount of proposal variability needed for the desired level of security depends on several system parameters,

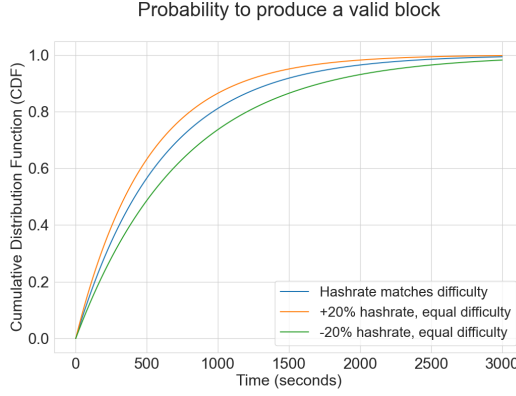


Fig. 3. Block proposal probability in Bitcoin

such as the number of miners and propagation delay. One method to satisfy BPP 2 is through sufficiently variable BT solution times, thus justifying the following property.

BTP 2 (Non-zero variability across BTs): Let us call an implementation I of the *Solve* function optimal if no other implementation asymptotically outperforms I . Given an arbitrary optimal I , the number of instructions I requires to solve a BT exhibits non-zero variability across different BTs of the same difficulty.

In Bitcoin, this property is satisfied due to the brute-force search of the cryptopuzzle solutions. Indeed, the distribution of times in Figure 3 shows significant variability. On the other hand, optimal algorithms to solve some classes of tasks terminate after a fixed number of operations. A simple example is matrix multiplication. An optimal algorithm for multiplying two matrices requires the same amount of computing steps regardless of the input and therefore does not satisfy BTP 2. BTP 2 is sufficient for fulfilling BPP 2, but not necessary. In general, BPP 2 may be satisfied by means other than variable task solution times. For example, it can be achieved by slightly varying the difficulty of BTs assigned to different block contexts. Additionally, in real systems, several factors such as network delay and heterogeneous computing power increase the variability in which nodes receive block proposals.

Block Proposal Property 3 (Adjustable lower threshold for difficulty). *When scaling up, the system efficiently and effectively controls a lower limit on the average time to propose valid blocks.*

In permissionless blockchains, new nodes may join the systems or, alternatively, existing nodes may get upgraded thus increasing the aggregate computing power of the network. Nonetheless, BPP 1 must be preserved when the computing power varies dynamically over time. Therefore, the system should be able to dynamically adjust the difficulty of creating new blocks.

BPP 3 is supported by the following **BTP 3 (Adjustable lower threshold)**: it is possible to efficiently and effectively assess and increase the difficulty of BTs produced by *Generate*.

In Bitcoin, for example, the BP rate is controlled by the cryptopuzzle's difficulty which is adjusted periodically every two weeks as follows. The network profiles the time to create blocks over the two-week period. If more than 2016 blocks are proposed in that period, the difficulty is raised; if fewer, the difficulty is lowered [119]. Thus the system adjusts the difficulty such that a block is proposed on average every 10 minutes, given the estimated total mining power.

Note that BPP 3 focuses on scaling up. We explicitly distinguish between adjustability for scaling up and scaling down, as the latter affects liveness rather than security.

Block Proposal Property 4 (Block Switchability). *A miner receives, on average, a higher reward by accepting a valid incoming block that would alter the local blockchain copy according to the protocol, and mining on top of it rather than by aiming to propose a conflicting block.*

In the Bitcoin core¹ protocol implementation, valid incoming blocks are appended to the local ledger and referenced in the following block. However, every miner is free to choose a different implementation so that miners should be incentivized to accept a valid incoming block and mine on top of it. Consider the case in which BTs are invariably solved after a fixed number of operations. Then, a miner close to producing a block may prefer to ignore incoming proposals and aim to produce conflicting blocks with the hope of winning the ensuing forks. Such a strategy increases branching, with the security consequences described in Section 3.2.

In essence, BPP 4 depends on the probability of existence of a valid block, on the average time for the miner to find a valid block, and on the probability of a proposed valid block to be included in the chain. There is a complex interplay between these three factors that may affect the miner's decision whether to switch. In general, if the difference in one factor is vast while the difference in the other two factors is slight, the differentiating factor will dominate the others. For example, in cases when the probability of existence of a valid block in the new context is zero, the miner may prefer to reject the incoming block and continue mining on top of the old one instead.

Ideally, the new block will be at least as good as the old block with respect to all three factors. This can be phrased as the following sufficient (but not necessary) conditions of BPP 4: (a) the probability of existence of a valid block with the new context is not reduced, (b) the expected time to propose a new block does not increase, and (c) the probability of a proposed valid block to be included into the chain does not decrease. Condition (c) is external to BTs. On the other hand, conditions (a) and (b) are supported by the following BTPs.

BTP 4a (No reduction in solvability): when switching to the new BT produced by *Generate* out of the new block context, the probability of existence of a valid solution for the new BT is not reduced compared to the BT produced by *Generate* out of the old block context. BTP 4a is necessary for condition (a) of BPP 4 since the probability of existence of a valid proposal relies on the probability of existence of the solution to the BT. BTP 4a is important for classes of tasks that may not have any valid solution and for which different BTs have a different probability of having a solution. For example, Bitcoin cryptopuzzles may not have any valid solution, but the probability for a cryptopuzzle to have a solution solely depends on its difficulty. Therefore, different cryptopuzzles of the same difficulty will have the same probability, thereby satisfying BTP 4a.

BTP 4b (Negligible BT generation time): the time to *Generate* a new BT is negligible compared to the time to *Solve* a BT. BTP 4b is related to condition (b) of BPP 4: a *Generate* of non-negligible duration with respect to the time required to solve a BT, directly increases the expected time to propose a new block.

BTP 4c (No increase in solution time): the expected time for the *Solve* function to find a valid solution for the new BT does not increase. This property compares the expected time to solve a new BT with the expected time to solve the old BT. BTP 4c directly affects condition (b) because an increased time to *Solve* the BT extends the expected time to propose a new block.

As mentioned above, a deterministic computation such as matrix multiplication is unfit in this regard because it is faster to complete an ongoing task rather than to start anew. BTP 4c is also important for classes of tasks that vary in terms of the expected time to find a solution. The property holds in Bitcoin because solving cryptopuzzles is a memoryless process. Hence the already performed computation does not affect the probability nor the expected time to find a solution. There is, however, one case where BTP 4c is violated in Bitcoin, namely when there is a difficulty

¹<https://github.com/bitcoin/bitcoin>

increase and the next block has a higher difficulty. In this case, the miners may prefer not to switch and rather continue mining on top of the previous block. Interestingly, there have been no reports of an increased rate of forks in Bitcoin at points where the difficulty increases.

Besides blocks, miners receive transactions from the network. Similarly to the arrival of a new block, the arrival of a new transaction being included in the block changes the block context and the BT. Intuitively, it should not be detrimental for miners to include new transactions in the block while mining; otherwise, miners may not be incentivized to modify the BT they are currently solving, leading to longer transaction confirmation delays and smaller total fees rewarded to miners. In Bitcoin, for example, it is a common practice for miners to replace transactions with lower fees as described in BIP125 [51]. However, in this work, we do not explicitly consider switchability due to incoming transactions as a property. Our focus is on the correctness requirements for blockchain, as defined in Section 2.4, as opposed to performance optimizations and metrics.

Block Proposal Property 5 (Block Proposal verification soundness). *The BP verification protocol has a high probability of rejecting an invalid BP i.e. the verification is sound.*

Every block proposal received from the network is first verified and then appended to the local copy of the ledger. In principle, the verification need not be deterministic. There exist several probabilistic verification protocols, such as zero-knowledge proofs [40]. Such types of verification have a probability of misclassifying invalid proposals as valid. If the probability of such false negatives is too high, many invalid blocks are included in the chain, negatively affecting the Chain Quality. Therefore, the probability of correct classification must be high, as stated in the following property.

BTP 5 (BT verification soundness): *Verify* has a high probability of rejecting an invalid BT solution.

In Bitcoin, the verification is deterministic: it suffices to verify that the block hash is lower than the target value. Therefore it is certain that a valid solution is classified as such.

3.3 Liveness Properties

Liveness in distributed systems has been extensively studied in several important works [83]. In the context of blockchain, liveness has been expressed in terms of Chain Growth and the rate at which blocks are appended to the ledger, as described in Section 2.4. In the following, we list a number of properties related to liveness: *Block Proposal verification efficiency*, *Block Proposal timeliness*, *Adjustable upper threshold for difficulty*, and *Block Proposal verification completeness*.

Block Proposal Property 6 (BP verification efficiency). *Given a BP, miners can efficiently verify its validity.*

Inefficient verification impacts the system in several negative ways. To begin with, inefficient verification slows down block propagation because miners validate a proposal before advertising it to the neighbors, as described in Section 2. Slower block propagation, in turn, delays the chain growth of honest nodes, possibly below the minimum allowed rate. In that sense, inefficient verification affects safety as well because slower block propagation increases the likelihood of branching. Besides, the need to invest non-negligible resources in verification may disincentivize the nodes from performing it. BPP 6 is supported by two BTPs.

BTP 6a (BT generation efficiency): *Generate* is efficient in creating a new BT. An efficient implementation of the generation primitive is central to block validation (as well as to other mechanisms) since *Generate* is executed as part of the preliminary computation to verify that the BT solved in the given BP is correctly derived from the block context of that BP. In reality, there are several factors affecting the efficiency of task generation. For example, if the BT generation

requires access to task metadata stored remotely, the generation time is directly affected by the fetching delay.

BTP 6a is similar to BTP 4b: both require BT generation to be efficient. However, BTP 6a is phrased as absolute efficiency in terms of the low runtime whereas BTP 4b stipulates relative efficiency compared to the solution time. In reality, it implies that BTP 6a is stronger than BTP 4b because the solution time is supposed to be significant according to BTP 1a. We nevertheless opt to introduce BTP 4b because there exist systems that satisfy BTP 4b but not necessarily BTP 6a, as we show in Section 6.

BTP 6b (BT verification efficiency): *Verify* is efficient. Clearly, an inefficient BT validation makes the BP validation inefficient as well. Thus, both BTP 6a and BTP 6b are necessary for BPP 6.

The exact meaning of efficiency depends on the specific blockchain system and the class of tasks being solved. It typically implies both low asymptotic computational difficulty and low computation times in practice since the verification is performed multiple times by a large number of nodes, including those with lower computing power.

In Bitcoin, the implementation of both *Generate* and *Verify* only consists of a small number of instructions. In particular, *Verify* entails a single hash computation and a comparison.

Block Proposal Property 7 (Block Proposal timeliness). *Given an arbitrary chain, the average time to produce a valid BP for the next block is “sufficiently fast”. Additionally, there is a very high probability of producing at least one valid BP for the next block within a time that is not “exorbitantly slow”. Both times are set so as to satisfy Chain Growth.*

The performance of blockchain systems is characterized by several important metrics related to liveness, namely the number of transactions appended per second and the time it takes for a transaction to become part of the main chain with high probability. Since transactions are batched and appended in blocks, these metrics are affected by the timeliness of BP. Without timely proposals, the performance of the system would be degraded, and the rate at which the chain grows may drop below the minimum rate required by the Chain Growth property.

It is actually non-trivial to capture the exact requirement of BP timeliness that is necessary for the blockchain system in practice. Intuitively, it is the opposite of hardness expressed as BPP 1: while all concurrent block proposals should be sufficiently hard, at least one concurrent block should be proposed within a reasonable time. However, probabilistic artifacts are not as detrimental to hardness as to timeliness. Suppose once in a rare while, there is a block that is secured by a relatively small amount of work. In that case, it will not be a significant problem for the security of the entire chain because later blocks will additionally secure the block. On the other hand, a block taking weeks to produce would cause a significant problem for Chain Growth and liveness in general. For this reason, BPP 7 specifies two probabilistic thresholds: one threshold to guarantee the average speed of appending blocks to the chain and another threshold to curb probabilistic artifacts. BPP 7 is supported by three BTPs. The first, **BTP 7a (BT generation efficiency)**, is equivalent to BTP 6a: *Generate* must produce new BTs efficiently. Since *Generate* is invoked every time a miner creates a BT and verifies a BP, an efficient implementation is important for the timeliness of BPs.

However, BTs may have no solution at all. For example, not all cryptopuzzles have a solution. A block cannot be secured by proof that the generated BT has no solution because neither cryptopuzzles nor most other task classes allow for such negative proof. This complication is addressed as follows in the Bitcoin design: *Solve* returns a special marker signifying that no solution has been found. In this case, the miner changes the block context, e.g., by modifying the timestamp or rearranging the transactions. Such a change results in a new block context and, consequently, a different BT for the miner to solve. This works, however, only if the following property is satisfied.

BTP 7b (BT solvability): a BT produced by *Generate* has a sufficiently high probability of having a solution. The probabilistic threshold may depend on the adjustable difficulty of the task (see BPP 8). In Bitcoin headers, for instance, the difficulty is represented as a 4-byte value, thus spanning between 0 and $4.3 \cdot 10^9$; even with a difficulty of 1, there is a 36.7% chance of not finding any valid hash in the entire 2^{32} nonce range. At the difficulty of one million, there is a 99.999905% that there is no solution in the entire nonce range [57].

Finally, BTs must allow for a sufficiently efficient implementation of *Solve*, which either finds a solution or concludes that no solution exists. It is conceivable, however, for a miner to use an implementation of *Solve* that may return without finding a solution even when a solution does exist. Imagine a class of BTs that consists of two categories: BTs that can be solved by a more efficient method and BTs that cannot. A miner may opt to use an implementation of *Solve* that only tries the more efficient method because it may be faster to apply the more efficient method to multiple generated BTs than to apply the more expensive method to a single BT. An efficient *Solve* implementation limits both the average runtime and the risk of probabilistic artifacts, as discussed above. A simple way to formulate this requirement is to consider the entire space of BTs producible by *Generate*, regardless of the block context:

BTP 7c (BT tractability): consider the entire set S of BTs that *Generate* may produce. Then, there exists an implementation A of *Solve* such that (a) A finds a solution for a “significant” fraction of BTs in S , (b) the average runtime of A over the BTs in S is “sufficiently fast” and (c) the fraction of BTs in S that takes A “exorbitantly high” time to process is very low. For cryptopuzzles, the best known *Solve* implementation is a brute-force search over the nonce space. This implementation always finds a solution if one exists. Its average runtime depends on the size of the nonce space (which is fixed in Bitcoin and equal to 32 bits) and on the probability of each point in the nonce space being a solution, which depends on the adjustable difficulty. The worst runtime for a single cryptopuzzle is simply the time required to scan the entire nonce space and calculate the SHA-256 hash function for each point in the space. However, the process of creating a block proposal may require solving multiple BTs until a BT with a solution is generated, as described above.

In Bitcoin, blocks are proposed every 10 minutes on average, whilst 99% of the blocks are found within 2763 seconds i.e. in less than five times the average. So far, the longest block proposal time has been for the first proposed block, i.e., after the genesis block. It took more than five days. At that time, however, Nakamoto was most likely the only miner in the system. The second most prolonged delay was one day, one hour, and eight minutes. It took place between blocks 15323 and 15324 [58]. On the other hand, if Bitcoin continues to operate for an extended time, a probabilistic artifact with a longer block creation time is likely to occur.

BTP 7c considers the entire set S of BTs that *Generate* may produce. In principle, a PoW system may consider the performance of *Solve* for a more restricted set of BTs. For example, given an arbitrary chain and a set of block contexts for the next block, the system may consider the set S of BTs that *Generate* produces for each block context. In this case, it might be possible to consider only the best runtime of a *Solve* implementation among all the tasks in this limited S . However, we are unaware of any research in the literature or any practical attempts to create such a PoW system.

Block Proposal Property 8 (Adjustable upper threshold for difficulty). *When scaling down, the system efficiently and effectively controls an upper limit on the average time to propose valid blocks.*

This property is the counterpart of BPP 3. When the number of blockchain participants decreases, the system should be able to adjust the difficulty to maintain the desired BP rate. Without this property, the time required to produce BP would increase, impacting BPP 7. At the task level, we define the following property. **BTP 8 (Adjustable upper threshold):** it is possible to efficiently and effectively assess and reduce the difficulty of BTs produced by *Generate*.

Block Proposal Property 9 (BP verification completeness). *The BP verification has a low probability of rejecting a valid BP i.e. the verification is complete.*

Considerations analogous to those presented for BPP 5 apply to BPP 9 as well. In the case of probabilistic verification, it should be highly unlikely to reject a valid solution; otherwise, miners may discard valid proposals, negatively affecting the BP rate and, therefore, Chain Growth. The BT property is stated as follows.

BTP 9 (BT verification completeness): The probability of *Verify* rejecting a valid PoC is low. In Bitcoin, the validation algorithm is deterministic, and therefore completeness is guaranteed.

3.4 Decentralization Property

It is critical that different blockchain nodes contribute blocks to the main chain in order to avoid centralization. This is achieved by incentivizing individual nodes to propose blocks through a system of rewards. However, rewards only work if each node has a fair chance of proposing a block that will be included in the main chain. In this section, we discuss the fairness of rewards due to block creation and how the process of solving tasks directly impacts these rewards.

Block Proposal Property 10 (BP fairness). *The probability of proposing a block is proportional to the relative amount of resources spent by the node on generating the BP.*

Assume that there are two miners, A and B , with a computing power of C_A and C_B , respectively. Further, assume that if A and B mine independently, their computing power grants them the respective probabilities of P_A and P_B to propose a block. Then, in order to disincentivize A and B from pooling their computing resources, the probability of A and B proposing a block if working together P_{AB} needs to be $P_{AB} \leq P_A + P_B$. In other words, the probability of proposing a block relative to the computational power must follow a non-superlinear function. Superlinearity is detrimental to decentralization because miners gain more than their proportional reward by joining forces or increasing their computational power.

In reality, systems supporting BPP 10 (such as Bitcoin) exhibit a linear increase in the probability. It is an open question whether a sublinear increase is desirable, as it implies that every miner is incentivized to split itself into as many miners as possible. At the BT level, BPP 10 is formulated as follows.

BTP 10a (No superlinear dependency on the resources): The probability that *Solve* produces a valid solution is proportional to the relative amount of resources spent by the node on executing *Solve*.

BTP 10a is satisfied by the cryptopuzzles in Bitcoin because (a) cryptopuzzles cannot be solved by any means other than brute force search, i.e., going through the entire space of possible solutions and checking each point individually, and (b) each point in the search space has an equal probability of being a solution. Properties (a) and (b) of cryptopuzzles are well-known [89], and result in a linear dependency on computational power, i.e. the rate at which solution candidates are checked is linearly proportional to the computational power.

Intuitively, the need for a random search over the solution space gives less powerful miners a chance to propose a block, i.e. to “get lucky.” The random search may start exploring a portion of the solution space already close to the task solution, allowing even a less powerful miner to solve the BT before more powerful ones do. If *Solve* implements a completely deterministic algorithm, the most powerful miner would always solve a BT before others, thus invalidating BPP 10.

It is not uncommon, however, to have partially random algorithms. This is the case when the search is guided by intelligent heuristics such as Genetic Algorithms and Simulated Annealing [33] where not every point in the search space has an equal probability of being a solution. In such situations, a more powerful miner can compute the heuristics and narrow the search more efficiently

than a resource-poor miner. For instance, in ML, such a powerful miner will likely be faster in computing the gradient over the training dataset. This may, in principle, lead to a superlinear dependency on the resource, but the exact impact depends on the specific class of BTs.

A second important requirement for proposal fairness is that experienced miners, i.e., miners that have been solving BTs for an extended period of time, do not get better chances compared to new miners. This has been formulated in BTP 1c (*Non-amortizability*); however, task amortizability is detrimental to fairness as well, and thus incorporated as **BTP 10b (*Non-amortizability*)**.

4 THE PROBLEM OF REPLACING CRYPTOPUZZLES WITH USEFUL TASKS

In this section, we discuss the problem of replacing cryptopuzzles with useful tasks. We base our discussion on the properties defined in Section 3, and on an extended architecture for blockchain systems obtained by modifying the architecture presented in Section 2.

The remainder of this section is organized as follows. Section 4.1 discusses different definitions of usefulness and related concepts. Next, Section 4.2 discusses the challenges of solving classes of tasks other than cryptopuzzles. Section 4.3 identifies different elements in Bitcoin-based blockchain platforms that have to be modified in order to support useful computation, while Section 4.4 discusses the impact of usefulness on BTPs. We then describe an extended architecture based on the Bitcoin model in Section 4.5.

4.1 Usefulness Definitions

While Section 2 presents computational tasks, there is no widely accepted definition of a *useful task*. Most of the works surveyed in Section 6, for example, include their own definition of usefulness. However, while the definitions differ in formulation, they broadly agree on the principle of a task solution having value external to the blockchain. An external party may be interested in solving one particular problem instance from a class of tasks. For example, a traveling agency may be interested in the TSP solution for a specific set of cities in which the agency operates. We call *supplier* the external party that provides the task consisting of one specific problem instance to be solved. We say that this type of usefulness is *strong* and is defined as follows.

Definition 4 (Strong usefulness). There is interest external to the blockchain system in solving a specific problem instance from a class of tasks.

Interest may be defined in different ways. One possibility is by financial means: if someone is willing to pay for the solution of a task, then the task is considered useful. On the other hand, there may be interest without payments. Consider, for instance, the many BOINC-based [16] volunteer computing projects. More broadly, a party may be interested in solving any problem instance within a given class of tasks. For example, suppose a scientist is interested in having a large dataset of TSP solutions for her research. We say that this type of usefulness is *weak* and is defined as follows.

Definition 5 (Weak usefulness). There is interest external to the blockchain system in solving any problem instance within a class of tasks.

Some proposals [17, 69] argue about the utility of weak usefulness. It is also possible to define an intermediate degree of usefulness between weak and strong. For example, a supplier may have constraints on the instance properties, e.g. TSP solutions for any random graph of exactly 1000 vertices.

In the case of weak usefulness, a supplier is not always necessary. A blockchain system can be designed so that problem instances of pre-defined classes of tasks can always be randomly generated and used as BT. We claim, however, that an external party is less likely to be interested in cryptopuzzle solutions compared to other computational tasks such as TSP, protein folding, or

ML. For this reason, it is imperative to extend the narrow scope of blockchain platforms solving cryptopuzzles to solving other classes of tasks.

4.2 Impact of Solving General Classes of Tasks on Task Properties

We now discuss the challenges of satisfying BTPs for general classes of tasks. As part of the system design, there must be agreement among solvers and suppliers about the class of tasks to solve. While it may theoretically be possible to devise a system that functions without a priori agreement on the class of tasks (e.g., the task may be encoded in a language that defines the objectives in addition to the input data), it complicates the design and challenges in a way non-instrumental to the contributions of this paper. Thus, we leave this case aside.

We observe that selecting a good class of tasks is essential because every property defined in Section 3 can be violated by a poor choice of task class. For example, finding Fibonacci numbers is amortizable, which violates BTP 1c (*Non-amortizability*). Matrix multiplications require the same number of operations, violating BTP 2 (*Non-zero variability across BTs*). Verifying that an optimization problem solution is optimal requires solving the task again, which is problematic for BTP 6b (*BT verification efficiency*). A deterministic solution algorithm may disincentivize solvers from adopting incoming blocks, as explained for BPP 4 in Section 3.2. Furthermore, a deterministic algorithm may pose a problem for BTP 4c (*No increase in solution time*) because a miner switching tasks “loses” the computation done so far. This affects block switchability, as miners close to producing a solution for the old block may prefer to do so rather than switch.

The difficulty of task classes needs to balance between BTP 1a (*BT hardness*) and BTP 7c (*BT tractability*). Realistically, the choice is between small-size instances of NP-hard classes and large-size instances of computational problems in P. Arguably, it is harder to adjust the difficulty of the former so that the latter may be better from the standpoint of BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*).

Compared to cryptopuzzles, typical classes of useful tasks stem from real-world problems, and modeling such problems as computational tasks results in much higher data storage requirements for the input data and solutions. For instance, the input to a TSP is a weighted graph, and the solution is a list of all vertices in the graph. The size of input and solution data thus grows proportionally to the size of the graph. This might be a practical issue if the tasks are stored in blocks. Since, in addition, tasks must be hard enough, the aggregated size of several TSP instances may result in a non-negligible storage space. ML is another good example of this issue since training data may be in the order of gigabytes, if not terabytes. Off-chain storage solutions tackle the problem; however, in that case, the data must be fetched first. This impacts the required bandwidth and all of the timeliness properties, such as task generation time, solution verification time, and the actual time to solve the task. In Section 5, we discuss how specific classes of tasks satisfy each BTP.

4.3 Usefulness in Bitcoin-based Blockchain Platforms

Adding support for usefulness to Bitcoin-based blockchain platforms relies on the design of three new elements: (i) *task supply model*, i.e. how task instances are provided to the system; (ii) *task selection policies*, i.e. how to select the next task instances that are to be solved at any given moment; and (iii) *incentive model* for solving useful tasks. This work focuses primarily on property analysis without providing a comprehensive analysis of the task supply model, selection policies, and incentive models. Nevertheless, we discuss possible directions below. Furthermore, Sections 6 and 8 describe different proposals found in the literature.

Regardless of how tasks are provided to the system, the system maintains a Task Supply State (TSS), which consists of a set of tasks that need to be solved or seeds used to generate the next task to solve. Similarly to the block context, the TSS is determined by the current state of the chain

and, thus, consistent across all nodes. Under strong usefulness, a supplier provides the tasks to solve. In the case of weak usefulness, tasks may be generated by the system itself according to a pre-agreed algorithm, which renders a supplier unnecessary. Alternatively, suppliers may be part of the weakly useful system and may determine classes of tasks to solve. Regardless of the type of usefulness or the existence of suppliers, sources of useful tasks may be classified as internal or external to the system.

A *task selection policy* determines how a node selects which task from the TSS to use as a BT for the next block. For example, a node may, in an uncoordinated fashion, freely choose any available task to solve. Another approach is to force all nodes to coordinate and attempt to solve the same task at a specific block height.

The third additional element is the *incentive model*. In a system without a supplier, it may be possible to retain the incentive model of Bitcoin based on transaction fees and a block reward. This possibility remains an open question, however, because solving useful tasks may require a different set of resources (e.g. in terms of hardware) and because a superlinear dependency on the number of node's resources (see BTP 10a) may disincentivize less resource-rich nodes from joining the system, thus requiring additional incentives.

Once a supplier is introduced, however, proposing a task must be subject to a fee. Such a fee not only makes sense from an economic perspective but is also required for system security and to mitigate DoS attacks. In fact, consider a simple incentive model consisting of modifying Bitcoin's rewards so that, all other things being equal, the coinbase is funded by the supplier's task proposal fee. In this scenario, a malicious supplier can easily propose an already-solved task and reap the transaction fees by colluding with a miner. Clearly, this is not desired.

4.4 Impact of Usefulness on Block Task Properties

Replacing cryptopuzzles with useful tasks creates friction with several BTPs. In the following, we describe those issues at an intuitive level. Section 5 goes into the details for three selected classes of tasks.

4.4.1 Impact on the Safety and Security Properties. While upholding BTP 1a (*BT hardness*) primarily depends on the choice of the class of tasks, the characteristics of specific instances still play an important role. In particular, some instances may adversely affect BTP 1a. For example, a TSP where cities lay in a simple geometric form such as a line or a circle is easier to solve than the general case. While submitted tasks can be tested against simple agreed-upon validity rules, it is infeasible to eliminate all simple instances in general. Since untrusted suppliers may submit arbitrary task instances, this becomes a challenge for BTP 1a. This challenge is exacerbated by the fact that a supplier may have a monetary incentive to collude with a solver, as described earlier. This effectively results in a *pre-computation attack* where the solver wins the race, gets the block reward, and splits the earnings with the supplier.

Furthermore, adjusting the difficulty requires an efficient mechanism to assess the hardness of an instance. Even when such a mechanism exists, adjusting the difficulty may not be feasible without violating strong usefulness. For example, adding cities to or removing cities from a TSP instance may, in principle, help adjust hardness. Yet, the resulting instance would be different. Therefore, strong usefulness poses a challenge for BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*).

Trivial approaches to replace cryptopuzzles with useful tasks present a clear trade-off between BTP 1b (*Context sensitivity*) and usefulness as well. On the one hand, BTP 1b stipulates that BTs must be context-dependent. On the other hand, usefulness implies that there is an external interest in a specific BT solution. Modifying the BT due to a context change invalidates strong usefulness

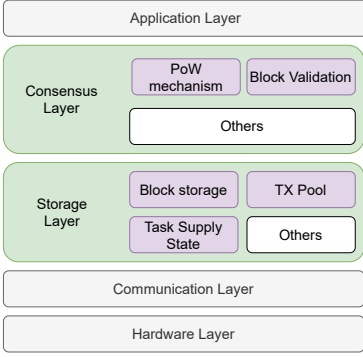


Fig. 4. Extended system architecture. Modified and newly introduced components are in a different color.

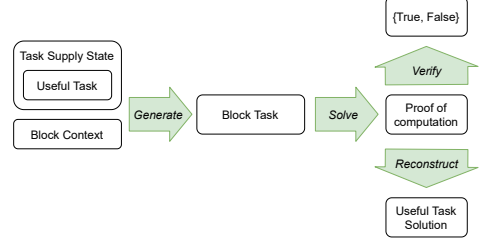


Fig. 5. Extended interface for useful tasks

in this case. Nevertheless, non-trivial approaches may succeed in replacing cryptopuzzles with a class of useful tasks while retaining both BTP 1b (*Context sensitivity*) and usefulness. Section 6.4 describes one example in detail.

4.4.2 Impact on the Liveness Properties. BTP 7b (*BT solvability*) stipulates that BTs should not be too hard to solve. However, suppliers may submit instances that are too hard to solve, mounting a DoS attack. The same challenges described for BTP 1a apply to BTP 7b.

Another issue to consider in practice is that there may be no instances available in the TSS. In such case, BTP 6a,7a (*BT generation efficiency*) does not hold true, and the system cannot progress. In a more extreme yet plausible scenario, an empty TSS leads to an inescapable deadlock if the TSS is updated through block-included transactions. In Bitcoin, on the other hand, BTs are randomly generated based only on the block context. Thus, Bitcoin miners have access to an inexhaustible supply of BT.

Usefulness requires “external interest” in the solution of a task, limiting the possible BTs – suppliers may be interested in a subset of all possible BTs. Mechanisms for supplying and/or restricting the BTs being solved by solvers are thus necessary in order to update the TSS.

4.4.3 Impact on the Decentralization Properties. As described in Section 3.4, any class of tasks whose best-known solution algorithm is deterministic is detrimental to BTP 10a (*No superlinear dependency on the resources*): it results in the most powerful solver being able always to solve BTs before others. Algorithms for solving useful tasks often consist of searches that get closer to the solution as they explore the solution space, giving faster solvers an advantage. In an extreme case, for instance, a few fast solvers may solve all tasks faster than all other solvers in the network, which may centralize block proposals, even though their computing power may be only slightly higher.

4.5 Extended Architecture and Interface to Support Useful Tasks

The Bitcoin architecture introduced in Section 2 does not support solving classes of tasks other than cryptopuzzles, nor does it include the design elements identified in Section 4.3. Hence, we introduce an extended architecture that solves these two deficiencies. The extension is based on fundamental design considerations as well as on the study of the existing approaches described in Section 8. The resulting architecture is used to discuss selected classes of useful tasks in Section 5 and proposed systems in Section 6. The roles of *clients* and *validators* remain the same as in Bitcoin;

Table 3. Extended Block Task Interface

	Generate	Solve	Verify	Reconstruct
Input	Task Supply State, Task Selection Policy, Block Context	Block Task	Block Task, Proof of Computation	Proof of Computation
Output	Block Task	Proof of Computation	Boolean	Task Solution
Descr.	Generates a Block Task based on the block content, the available tasks, and the task selection policies.	Solve the Block Task and outputs a Proof of Computation.	Verify that the Proof of Computation is a valid solution to the Block Task.	Extract the BT solution from the Proof of Computation.

miners, however, are renamed to *solvers* so to reflect the new responsibility: solving tasks. The role of *supplier*, as already extensively described in Section 4.1, is introduced as well.

We start by expanding the scope of several system components, as shown in Figure 4. Transactions in the TX Pool now include all metadata necessary to support the new task supply models. Therefore the modified TX Pool supports the storage of new transaction types to, for example, advertise a task. The TSS introduced in Section 4.3 constitutes a separate component. The Block storage stores all proposed blocks and the relative metadata. In practice, large tasks may lead to blocks of large sizes. In those cases, storing a portion of this data off-chain may be necessary.

The Block validation component contains the extended primitives to check the validity of an incoming block. Similarly, the PoW mechanism component implements all primitives related to computing the solution of useful tasks. We are now ready to describe an extended interface to support the modified system architecture.

Compared to the base interface, *Generate* now takes as input the TSS in addition to the block context. Solvers generate the BT from one of the currently available instances in the TSS, chosen according to the task selection policy. The system may enforce a specific task selection policy, e.g., the earliest unsolved instance included in the chain, or leave the selection up to the individual miner. A simple implementation of *Generate* may output a BT from the TSS without any modifications. A more elaborate construction may, e.g., transform the coordinate reference system [28] of a TSP instance or, more generally, transform the task domain. The idea behind such a transformation is that the derived task domain may lend itself better to satisfying the BT properties. In such cases, a reverse transformation has to be applied to the solution produced by *Solve* in order to reconstruct a solution to the original task, as given by the supplier. The extended interface supports those cases.

Some of the works analyzed in Section 6 use a specific BT generation algorithm to derive the BT out of the context. Other works opt for a task prioritization algorithm and tackle BTP 1b with other means, such as shuffling of tasks (see Section 6.5).

The semantic and input/output of the extended *Solve* and *Verify* remain equivalent to those in the base interface. We note that most of the works analyzed in Section 6 do not assume any constraint on the *Solve*, which implies leaving the freedom to solvers to select any solution algorithm.

When a transformation is performed by *Generate*, *Reconstruct* is applied to the PoC produced by *Solve* in order to obtain the solution of the original task. Figure 5 shows the interactions between different operations of the extended interface.

5 SELECTED CLASSES OF USEFUL TASKS

This section surveys some of the challenges in replacing cryptopuzzles with useful tasks. The challenge discussion is based on the property framework defined in Section 3. We start by identifying the general trade-offs related to optimization and decision tasks. Then, we discuss salient aspects of BTP support of the three classes of useful tasks introduced in Section 2, namely k-OV, TSP, and

ML. They represent the most common classes of tasks that have been considered in the literature, thus providing meaningful insights into the practical challenges of designing useful PoW systems. We have found that all these three classes satisfy BTP 6a,7a (*BT generation efficiency*) and BTP 4b (*Negligible BT generation time*). On the other hand, for the reasons illustrated in Section 4.4.1, BTP 1b (*Context sensitivity*) and BTP 4c (*No increase in solution time*) are not upheld by any of the classes. The analysis is summarized in Table 4.

5.1 Decision and Optimization Tasks

Solving optimization problems means finding an optimal solution. However, verifying that a solution is optimal is typically as difficult as finding an optimal solution in the first place [2]. This applies in practice regardless of the advances in the theory of combinatorial optimization and in Verifiable Computing (VC). VC provides proof of computational integrity without any assumptions on hardware or failures [67].² A VC proof of a Concorde execution would be, in theory, sufficient to verify that the TSP solution is optimal. However, the time overhead to create an efficiently verifiable proof for arbitrarily complex types of operations is currently prohibitive. For instance, a state-of-the-art algorithm proving a simple matrix multiplication of size 512 takes an order of minutes to execute on commodity hardware as observed in [67]. This is significantly longer than performing the multiplication itself but it can be verified faster. In summary, as of today, achieving both BTP 7c (*BT tractability*) and BTP 6b (*BT verification efficiency*) at the same time remains an open challenge for optimization tasks.

A possible relaxation for optimization problems is to accept suboptimal solutions that lie within a fixed margin from the optimal. In some cases, it is possible to efficiently estimate a bound on the optimal value of the instance. For example, the Christofides algorithm [26] is an approximation algorithm for TSP, which is efficient and proven to provide a solution at most $3/2$ worse than the optimal solution [26]. It would be enticing to use this method to derive a validity threshold for optimization problems. The challenge, however, is that approximation algorithms like Christofides do not reliably produce solutions at a fixed factor i.e. they may compute a solution much better than $3/2$ of the optimal. Consider the case in which the Christofides algorithm incidentally computed an optimal solution. Clearly, it would be impossible to produce a better solution than that, had this been the scheme to derive the margin. In this situation, solvers cannot produce a valid solution, thus violating BTP 7a (*BT generation efficiency*). Because of those issues, especially with regard to BTP 6b, the use of optimization problems as cryptopuzzle replacements leaves some important open problems.

On the other hand, decision problems are verifiable in polynomial time. However, the main challenge for decision problems is determining a suitable threshold: a too-pessimistic threshold makes the task easy to solve, violating BTP 1a (*BT hardness*). In contrast, a too-optimistic threshold may take a long time to produce a valid solution, violating BTP 7c (*BT tractability*). In the worst case, the threshold might be so high (or low) that there exists no feasible solution for the BT, which violates BTP 7b (*BT solvability*). This applies to TSP and DL tasks discussed respectively in Section 5.3 and Section 5.4. Thus, the choice of the threshold and how it is calculated (or supplied) is central to the design of the solution.

5.2 Characterization of k-OV as a Block Task

In the following, we characterize a blockchain based on k-OV tasks where *Solve* determines which vectors are orthogonal by solving Equation 1 and the PoC contains OV sets. While we focus on the case of $k = 2$, the characterization also applies to other values of k .

²For further details on VC, we refer the interested reader to [31, 70, 94, 116].

Table 4. Block Task Properties for different classes of tasks

	BTP	Interface	k-OV	TSP	DL
Safety	BT hardness	<i>Generate</i>	Polynomial complexity requires infeasible large-scale input requirements	NP-complete but vulnerable to simple special cases	Dependent on the interplay between validity threshold, training data, and NN parameters
	Ctx sensitivity	<i>Generate</i>	Trade-off with strong usefulness		
	Adjustable lower threshold	<i>Generate</i>	Tasks can be batched into harder BTs		
	No reduction in solvability	<i>Generate</i>	k-OV BTs always admit a valid solution	TSP tasks always admit a valid solution	Dependent on the interplay between validity threshold, training data, and NN parameters.
	Negligible BT generation time	<i>Generate</i>	BTs are generated in constant time		
	Non-amortizability	<i>Solve</i>	Conjectured to be difficult to amortize effectively	Not explored in any related work	Invalidated by transfer learning and similar techniques
	Non-zero variability across BTs	<i>Solve</i>	Commonly used solution algorithms are characterized by low variability	Variability orders of magnitude smaller compared to cryptopuzzles. Likely trade-off with <i>BT hardness</i>	Unclear whether sufficient due to the multitude of SGD variants
	No increase in solution time	<i>Solve</i>	Not satisfied		
	BT verification soundness	<i>Verify</i>	Set inclusion is verifiable in constant time	Trade-off with BTP 6b (<i>BT verification efficiency</i>)	Satisfied by the threshold-based approach
Liveness	BT generation efficiency	<i>Generate</i>	BTs are generated in constant time		
	BT solvability	<i>Generate</i>	BTs always admit a valid solution		
	Adjustable upper threshold	<i>Generate</i>	Tasks can be partitioned into easier BTs	Trade-off with strong usefulness	Function of the validity threshold and training data. Opportunities to investigate the impact of NN parameters
	BT tractability	<i>Solve</i>	Achievable by partitioning BTs into easier sub-tasks	Possibly vulnerable to hard special cases targeting specific algorithms	Dependent on the interplay between validity threshold, training data and NN parameters
	BT verification efficiency	<i>Verify</i>	Trade-off with BTP 9 (<i>BT verification completeness</i>)	Trade-off with BTP 5 (<i>BT verification soundness</i>) and BTP 9 (<i>BT verification completeness</i>)	BTs are verified in polynomial time. Bandwidth may hinder efficiency
	BT verification completeness	<i>Verify</i>	Trade-off with BTP 6b (<i>BT verification efficiency</i>)		
	BT verification completeness	<i>Verify</i>	Trade-off with BTP 6b (<i>BT verification efficiency</i>)		
Detr.	No superlinear dependency on the resources	<i>Solve</i>	Conjectured as not satisfied		

A straightforward implementation of Equation 1 computes n^2d multiplications. Therefore, given a computing power c measured in operations per second and a time budget t , assuming square matrices such that $d = n$, the upper bound on the size of the matrices n solved within the time budget is $(ct)^{1/3}$. As mentioned in Section 2.1, even the best-known algorithm iterates over all vector permutations. However, OV complexity is insufficient to satisfy BTP 1a due to being polynomial. To illustrate the point, we perform a back-of-the-envelope calculation. The entire calculation can be found in Appendix A but our assessment is that a mining pool of 400 PFLOPS requires matrices of dimension $n \approx 6.2 \cdot 10^6$ to solve a k-OV instance in 600 seconds. If every vector dimension requires 1 bit of representation, each matrix of that size (n^2) would occupy 4.8 TB. Since every block must include a BT, we conclude that the space requirement of a k-OV secured blockchain is impractical.

All k-OV BTs admit a valid solution, as required by BTP 7b (*BT solvability*) and BTP 4a (*No reduction in solvability*) because it is always possible to produce the list of OVs given enough time. A

task can easily be batched with another task or decomposed into sub-problems by partitioning the input matrices, thus satisfying BTP 7b. Task batching facilitates BTP 3 (*Adjustable lower threshold*), while task partitioning helps with BTP 8 (*Adjustable upper threshold*) and BTP 7c (*BT tractability*).

In theory, k-OV is trivially amortizable if different BTs include copies of the same vector, which is detrimental for BTP 1c (*Non-amortizability*). In practice, the effectiveness of amortization over multiple BTs is constrained by the fact that computing multiplications and sums is a fairly cheap operation whose complexity mostly comes from the large input scale, as explained above.

On the other hand, a limit of k-OV is that solution algorithms are deterministic and characterized by low variability. While likely non-zero, thanks to hardware heterogeneity, this may be insufficient for BTP 2 (*Non-zero variability across BTs*). In addition, as anticipated in the space requirement estimates, solvers with higher FLOPS are likely to solve k-OV BT faster than other solvers due to such a deterministic solution search. This is clearly detrimental for BTP 10a (*No superlinear dependency on the resources*).

BTP 5 (*BT verification soundness*) is satisfied and efficient to compute because verifying that a vector in the PoC is present in the input instance takes constant time. BTP 9 (*BT verification completeness*), on the other hand, presents a trade-off with BTP 6b (*BT verification efficiency*) since a PoC may not include all the existing OV. Therefore, the only way to verify the PoC is to re-execute the calculation at the expense of BTP 6b.

5.3 Characterization of TSP as a Block Task

We now characterize a blockchain based on TSP tasks. An instance in the TSS consists of an arbitrary set of cities. *Solve* finds a valid permutation of cities which minimizes the cost as expressed by Equation 2. Such permutation represents the PoC, and solvers are free to choose their preferred solution algorithm.

As mentioned in Section 4.4, simple tasks jeopardize BTP 1a (*BT hardness*). Despite TSP being NP-complete, it is extremely fast to solve small instances: it takes Concorde 10 seconds on average to find the optimal solution to TSPLIB instances with fewer than 1000 cities [56]. Heuristic algorithms are even faster. Enforcing a minimum size on the instances, e.g. to be above 5000 cities, does not solve the issue due to the abundance of easy-to-solve special cases. Similarly, there exist particularly difficult instances that pose a problem for BTP 7c (*BT tractability*). For example, the authors of [56] present a family of instances for which Concorde takes 10^6 longer execution times compared to instances of similar size. At any rate, to the best of our knowledge, hard instances have been devised for individual algorithms; hence, using alternative solution algorithms should mitigate the problem.

Unlike k-OV tasks, TSP tasks cannot be partitioned. Hence, BTP 8 (*Adjustable upper threshold*) poses a challenge, as previously discussed in Section 4.4.1. BTP 3 (*Adjustable lower threshold*), on the other hand, may still be addressed by batching multiple BTs together. Since a solution to the optimization task is guaranteed to exist, BTP 7b (*BT solvability*) and BTP 4a (*No reduction in solvability*) are both satisfied.

The authors of [56] show that multiple executions of Concorde on TSPLIB instances have different runtimes, spanning roughly one order of magnitude and centered on the average runtime. For example, an instance solved after 50 seconds on average has executions that terminate after 10 as well as 100 seconds. This result is fairly stable across the benchmarked instances of different sizes and seems promising for the support of BTP 2 (*Non-zero variability across BTs*). However, the desired degree of variability is not precisely known: instances up to 200 cities terminate in a fraction of a second, and variability of a few seconds may be insufficient to support BTP 2. In Bitcoin, assuming the current level of cryptopuzzle difficulty, an ASIC computing 10^{14} hashes/s, such as the one described previously, has an expected time of 1157 days to propose a block and a standard

deviation of 10^4 days, based on the formula in [102]. Therefore, to equal Bitcoin’s variability, a TSP BT should terminate in about 1000 days, which is at odds with BTP 1a; faster executions may not have the required variability.

To the best of our knowledge, the question of whether and to what extent TSP tasks support BTP 1c,10b (*Non-amortizability*) has not been explored in any related work and remains open.

BTP 6b (*BT verification efficiency*) is a challenge for optimization tasks, as discussed in Section 5.1. Inefficient verification impacts BTP 5 (*BT verification soundness*) and BTP 9 (*BT verification completeness*) since the decision to accept or reject a solution claimed as optimal requires its computation in the first place.

We note that many state-of-the-art TSP solution algorithms follow stochastic approaches. It is, however, unclear to which degree BTP 10a (*No superlinear dependency on the resources*) is satisfied. The analysis is algorithm-specific, and so far, this metric has not been of interest in the analysis of algorithms, which has rather focused on other performance-related metrics. We conjecture that most solution algorithms for TSP do not satisfy BTP 10a (*No superlinear dependency on the resources*).

5.4 Characterization of DL as a potential area for Block Tasks

DL is a promising candidate as a class of tasks because it presents a high degree of customization. For instance, DNN may have varying layers, arrangement of neurons, activation functions, etc. Those parameters can be constrained (e.g., the number of neurons must not exceed a specific threshold) and are efficiently verifiable by inspecting the DNN. Nonetheless, other parameters, such as those related to training (number of training batches, type of SGD, and so on) are not efficiently verifiable by direct inspection. In the following, we characterize blockchain based on DL as BTs.

The TSS comprises supplier-provided datasets and corresponding error thresholds thereafter referred to as validity thresholds. *Solve* updates the set of weights according to Equation 4, thus minimizing the error in Equation 3. The resulting DNN is used as PoC. Note that in the ML community, the error value on training data is a proxy goal for achieving good generalization performance with previously unseen data. Zero error on training data is usually a sign of overfitting, and many mechanisms, such as regularization techniques, have been developed to avoid that [113]. Thus, in the following, we consider a solution valid if it is above a certain arbitrary non-zero error threshold. However, the error should not be too big either. *Verify* checks that the PoC satisfies such error requirements. Using a validity threshold makes some of the observations in Section 5.3 applicable to DL BTs as well.

Additionally, there are specific constraints due to the nature of DL tasks. For instance, a small dataset constituted by a handful of images is quickly and efficiently learned by a DNN, thus invalidating BTP 1a (*BT hardness*). Hence, the dataset should have sufficient variety, complexity, and size. Similarly, it has been shown that sufficiently over-specified DNNs (i.e. DNNs that are larger than needed in relation to the task at hand) are easy to train because global optima are ubiquitous and generally computationally easy to find [81]. Therefore, in the general case, BTP 1a depends on a non-trivial interplay between (a) validity threshold, (b) training data, and (c) DNN parameters. These parameters also impact BTP 4a (*No reduction in solvability*), BTP 7c (*BT tractability*), and even BTP 7b (*BT solvability*) since a poor choice may result in the absence of a solution. We believe that calibrating these parameters effectively and on a per-instance basis remains an open challenge.

On the positive side, these parameters result in a high degree of customization inherent to DL tasks, which facilitates the implementation of BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*). Tinkering with the training data invalidates strong usefulness. Depending on the external interest, altering the validity threshold may also invalidate strong usefulness. On

the other hand, adapting the DNN parameters may be a viable strategy due to the ability to characterize their impact on the training time [66]. However, additional work is needed to generalize across different DNN, data sizes and hardware configurations. We regard this as a research gap and describe it in Section 7.

In ML, it is a common practice, known as transfer learning, to train a DNN on a dataset and apply the trained DNN to a different but related problem to reduce the training time [59]. Transfer learning invalidates BTP 1c,10b (*Non-amortizability*) by amortizing the training time with past computation. Therefore, a blockchain replacing cryptopuzzles with DL tasks should neutralize the training time reduction due to transfer learning, which is instead a desideratum in usual ML applications.

BTP 2 (*Non-zero variability across BTs*) depends on the training algorithm. SGD, along with its variants, is a popular choice for optimizing DNN. SDG is characterized by significant variability. However, some works have criticized SGD for its slow asymptotic convergence due to its stochastic nature [65] and proposed modifications to reduce the variance [65, 104]. Therefore, it remains unclear whether the cutting-edge algorithms exhibit sufficient variability.

Verify may require the transfer and processing of large amounts of data, potentially invalidating BTP 6b (*BT verification efficiency*). However, once the necessary data is available locally, *Verify* takes polynomial time by requiring a forward pass for each data point. In addition, the threshold-based approach guarantees BTP 5 (*BT verification soundness*) and BTP 9 (*BT verification completeness*).

It is well-known that increasing the number of GPUs dedicated to ML training shortens the training time, in some cases up to two orders of magnitude [125]. In addition, the authors of [10] show that memory bandwidth is the second-order bottleneck in training time after GPU computing power. Therefore, miners that invest heavily in more GPUs with higher memory bandwidth are measurably faster to *Solve* a BT. Following this observation, we conclude that BTP 10a (*No superlinear dependency on the resources*) does not hold for DL and that the most powerful miners are likely to propose more than their fair share of blocks.

6 EXISTING PROPOSALS FOR SUPPORTING PROOF OF USEFUL WORK

Several system designs have been proposed to solve the problem of replacing cryptopuzzles with useful tasks. This section analyzes a representative subset of these proposals, their BTP support, as well as how each proposal positions itself wrt. usefulness, supply models, task selection policies, and incentives. In short, none of the analyzed proposals comprehensively satisfies all the BTPs. We highlight the trade-offs of each solution and conclude that, for the time being, replacing cryptopuzzles with useful tasks remains an open problem.

In the face of the challenge, some of the proposals only introduce a partial replacement: they combine cryptopuzzles with solving useful tasks in a variety of ways. For each such “hybrid” system, we elaborate on the balance between useful computation and solving cryptopuzzles.

Tables 5 and 6 provide an analysis overview. Each cell has been marked either as *positive proof*, *positive conjecture*, *not conjectured*, *negative conjecture*, or *negative proof*. A positive (or negative) proof, denoted as $[+]$ (or $[-]$), means that there exists formal or empirical proof to substantiate (or refute) the validity of the property. A positive (negative) conjecture tag, denoted as $[\pm]$ ($[\mp]$), refers to conjectures based on intuitive, logical reasoning. Some of these conjectures are found in the literature, while others are introduced in our analysis. However, these conjectures have not been proven theoretically or corroborated experimentally. Not conjectured (symbol $[=]$) means that there is no evidence or even conjecture one way or the other, regardless of whether the property is discussed in the proposed system. This happens when it is unclear how to approach reasoning in the first place, or there exist opposite arguments leading to an inconclusive outcome.

By extensive and recursive literature review, we identified 36 proposed blockchain systems based on PoW in which nodes solve useful tasks as part of the block proposal process. The count includes systems that avoid solving cryptopuzzles altogether, as well as hybrid systems where the nodes solve both cryptopuzzles and useful tasks.

In this section, we analyze those systems that provide sufficient technical detail in their description to assess BTP support and that assume the same trust and synchronization model as Bitcoin. If either of those models significantly departs from those of Bitcoin, it affects the consideration of challenges to a great extent. We exclude those systems from the analysis in this section, but we mention them in Section 8. Some of the works studied below exhibit significant similarities. Thus, we divide the works into six affinity groups: Cryptopuzzle equivalence, Threshold-based tasks, Complementing useful tasks with staking, Domain transformation, Shuffling of tasks, and Solution search while mining. For each group, we analyze one or two systems in detail.

We found that BTP 6a,7a (*BT generation efficiency*) and BTP 4b (*Negligible BT generation time*) are satisfied in all the systems analyzed in the current section and, therefore, do not repeat this result for each individual system.

Table 7 summarizes the analysis of usefulness and supply models, task selection policies, and incentives of those proposals.

6.1 Attempts at cryptopuzzle equivalence

As the high hurdle of the challenge of replacing cryptopuzzles was unfolding over the years, later proposals came up with sophisticated techniques that were trying to compensate for the property gap between cryptopuzzles and other classes of computational tasks. On the other hand, earlier proposals attempted to carefully craft a dedicated class of tasks that would be useful, and yet would possess properties similar to those of cryptopuzzles. Among those works are Primecoin [69] and Gapcoin [42]. While motivating and inspiring later ideas, the crafted classes were criticized for the lack of practical applications, which limited their usefulness. Besides, not all requirements were analyzed or even known. Hence, despite being carefully crafted, the tasks do not satisfy all of the BTPs we list in Section 3.

Primecoin was introduced in 2013 and is considered the first practical attempt to replace cryptopuzzles with useful computation. In Primecoin, solvers find long sequences of prime numbers, also known as chains. Primecoin accepts three types of chains. In the first type, called Cunningham chain [49] of the first kind, each prime number (except the first) must be equal to twice the previous prime number minus one, for example, 6121, 12241, and 24481. In the second type, called Cunningham chain of the second kind, each prime must equal twice the previous prime plus one. The last type combines the previous two and is called bi-twin; a bi-twin chain is a chain where primes form pairs that are two units apart from each other and where the average of each pair is double the average of the previous pair. An example of bi-twin chain is 211049, 211051, 422099, and 422101. For all three chain types, it is possible to define an *origin* i.e. a number close to the first element of the prime chain. In the numerical examples provided earlier, the two origins are respectively 6120 and 211050. Primecoin makes the work context-dependent by requiring the origin of the prime chain to be divisible by the block header hash. Therefore, miners first assemble a block proposal and then start searching for valid prime chains.

Primecoin uses probabilistic primality tests, namely the Fermat test [121] and the Euler-Lagrange-Lifchitz test [76], to verify the validity of a block. Those tests check whether the equality in Fermat's little theorem [52] holds for a set of randomly chosen values. The confidence that a number is prime increases with the number of tested values that satisfy the equality.

Generally, the longer the prime chain, the harder it is to find one. However, since a chain longer by just one prime number may be considerably harder to find, Primecoin accepts fractional chain

Table 5. Support for security and safety Block Task Properties in surveyed systems

Paper	Limited BP rate	Non-zero BP variability	Adjustable lower threshold	Block switchability	BP verification soundness
Primecoin [69]	(a) $[\pm]$ Hardness of searching prime chains. (b) $[+]$ Context depends on the prime chain origin. (c) $[-]$ Unclear amortizability.	$[-]$ Unclear variability across BTs.	$[+]$ The required prime chain length is adjustable up to fractional length.	(a) $[+]$ Every task has a solution. (b) $[+]$ BT generation time is negligible compared to finding a solution. (c) $[-]$ Unclear increase in solution time.	$[\pm]$ Very high probability that invalid solutions are rejected.
DLchain [24]	(a) $[\mp]$ Short-term targets are unilaterally decided by the supplier. (b) $[+]$ Context depends on the training seed. (c) $[-]$ Unclear non-amortizability for proposed DL tasks.	$[\pm]$ Improving the accuracy up to the target is a random trial and error process.	$[-]$ The effectiveness of adjustability through short-term targets remains unclear.	(a) $[\mp]$ The probability that a valid solution exists may be reduced. (b) $[+]$ BT generation time is negligible compared to finding a solution. (c) $[\pm]$ Reaching the target is a random trial and error process.	$[+]$ Models having unsatisfactory accuracy are always rejected by validators.
Coin.AI [6]	(a) $[-]$ Unclear hardness due to underspecified validity threshold and context-free grammar properties. (b) $[+]$ Model initialization depends on block context. (c) $[-]$ Unclear non-amortizability for generated NN tasks.	$[\pm]$ Variability is likely provided by heterogeneous NN training times.	$[\mp]$ No approach is described to increase hardness in the case of simple instances.	(a) $[+]$ Valid solutions to tasks exist thanks to the difficulty decay. (b) $[+]$ BT generation time is negligible compared to finding a solution. (c) $[-]$ Changing context increases the expected time to find a solution.	$[\mp]$ Because of the difficulty decay, unsynchronized miners may accept invalid solutions.
Hybrid Mining [21]	(a) $[-]$ No control over the hardness of user-supplied tasks. (b) $[-]$ Useful tasks are not context-sensitive. (c) $[-]$ Unclear non-amortizability of NP-complete tasks.	$[-]$ Unclear BT solution time variability for useful tasks.	$[-]$ Adjustability of useful tasks is not discussed and remains unclear.	(a) $[-]$ The probability that a valid solution exists depends on the class of tasks. (b) $[+]$ BT generation time is negligible compared to finding a solution. (c) $[-]$ The expected solution time depends on the class of tasks.	$[-]$ Not discussed. Verification soundness depends on the type of NP-complete task.
Proofs of Useful Work [8]	(a) $[\pm]$ BT proved hard on average. Unclear task storage requirements. (b) $[+]$ Nonce-dependent context. (c) $[+]$ Polynomial evaluations are proved to be not amortizable.	$[\mp]$ Similar number of operations for same difficulty tasks.	$[\mp]$ Trade-off with strong usefulness.	(a) $[+]$ Every BT has a solution. (b) $[+]$ BT generation time is negligible compared to finding a solution. (c) $[-]$ The expected time to propose a new block increases.	$[+]$ The property is proven by the authors.
Proof of eXercise [109]	(a) $[\pm]$ High dimension input size coupled with a Proof of Hardness. (b) $[\pm]$ Assigning miners to the task depends on the context. (c) $[\mp]$ Matrix operations are likely amortizable.	$[-]$ Unclear variability across BTs.	$[\pm]$ Batching of tasks to regulate the proposal generation time.	(a) $[-]$ Depends on the class of tasks. (b) $[+]$ BT generation time is negligible compared to finding a solution. (c) $[-]$ The expected time to propose a new block increases.	$[\pm]$ Invalid proposals are likely to be rejected, proportionally to the number of verifications performed.
Proof of Search [108]	(a) $[+]$ Hardness follows from cryptopuzzles. (b) $[+]$ Context sensitivity guarantees are similar to Bitcoin. (c) $[+]$ Non-amortizability follows from cryptopuzzles.	$[+]$ Follows from cryptopuzzles.	$[+]$ By increasing the number of leading zeros of valid blocks.	(a) $[\mp]$ Switching to a new instance may reduce solvability. (b) $[+]$ BT generation time is negligible compared to finding a solution. (c) $[\mp]$ Switching to a new instance may increase solution time.	$[+]$ Invalid solutions are always rejected.
Ofelimos [41]	(a) $[\pm]$ Hardness follows from cryptopuzzles and DPLS. (b) $[+]$ Cryptopuzzles provide context dependency. (c) $[\pm]$ Non-amortizability of BT provided by cryptopuzzles.	$[+]$ Variability follows from cryptopuzzles and DPLS.	$[\pm]$ It may be possible to increase the difficulty by adjusting T_2 under certain assumptions on DPLS.	(a) $[\pm]$ Solvability of new BTs is not reduced. (b) $[+]$ BT generation time is negligible compared to finding a solution. (c) $[\pm]$ Solution time is likely not to increase for constant values of T_2 .	$[\pm]$ Trivial for cryptopuzzles. Derived from SNARGs for useful tasks.

lengths e.g. 11.6, where the decimal part is the Fermat test remainder. Primecoin uses such a remainder to approximate a linear difficulty function.

Since the tasks are not user-supplied and the system accepts any prime chain of sufficient length, we classify such a class of tasks as weakly useful. For the same reason, there is no task selection policy. Miners allocate all their computing steps in finding a Cunningham chain of sufficient length. Like in Bitcoin, Primecoin miners receive block rewards and transaction inclusion fees.

BTP 1a (*BT hardness*) and BTP 7c (*BT tractability*) stem from an appropriately chosen length of the prime chain, since a longer chain is harder to find. Such length is adjustable up to fractional values, supporting both BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*). BTP 1b (*Context sensitivity*) is guaranteed by tying the block to the origin of a prime chain. BTP 1c,10b

Table 6. Support for liveness and decentralization Block Task Properties in surveyed systems

Paper	BP Verification efficiency	BP Timeliness	Adjustable upper threshold	BP Verification completeness	Proposal Fairness
Primecoin [69]	(a) [+] See BTP 7a. (b) [+] Fermat-based primality tests are efficient.	(a) [+] BT generation is efficient. (b) [+] Each BT has a solution. (c) [±] Length of prime chain empirically chosen to be tractable.	[+] The required prime chain length is adjustable up to fractional length.	[+] Valid solutions are always accepted.	(a) [=] Unclear dependency on the employed resources. (b) [=] See BTP 1c.
DLchain [24]	(a) [+] See BTP 7a. (b) [±] Efficient verification in the absence of forks. Inefficient otherwise.	(a) [+] <i>Generate</i> in constant time. (b) [±] Short-term targets are unilaterally decided by suppliers. It may not be solvable. (c) [±] Short-term targets are unilaterally decided by suppliers. It may not be tractable.	[=] The effectiveness of adjustability through short-term targets remains unclear.	[+] Validators always accept models having satisfactory accuracy.	(a) [±] Improving the accuracy up to the target is a random trial and error. (b) [=] See BTP 1c.
CoinAI [6]	(a) [+] See BTP 7a. (b) [±] Validation requires a forward pass of the neural network and a check of the hyperparameter mapping function.	(a) [+] <i>Generate</i> takes constant time. (b) [±] Tasks are solvable thanks to the difficulty decay. (c) [=] Unclear tractability due to under-specified validity threshold and context-free grammar properties.	[±] Threshold decay makes instances easier to solve.	[±] Because of the difficulty decay, unsynchronized miners may reject valid solutions.	(a) [±] Not discussed. The probability to solve ML instances likely grows superlinearly with the resources employed. (b) [=] See BTP 1c.
Hybrid Mining [21]	(a) [+] See BTP 7a. (b) [=] Not discussed. Useful tasks may be inefficient to verify.	(a) [+] <i>Generate</i> runs in constant time for both cryptopuzzles and useful tasks. (b) [±] Useful tasks may not have valid solutions. (c) [±] Tasks without valid solutions cannot be solved fast enough.	[=] Adjustability of useful tasks is not discussed and remains unclear.	[=] Verification completeness of tasks is not discussed.	(a) [±] Useful task solution probability is likely to grow superlinearly with the resources employed. (b) [=] See BTP 1c.
Proofs of Useful Work [8]	(a) [+] See BTP 7a. (b) [±] The verification efficiency is proved to be linear.	(a) [+] <i>Generate</i> runs in linear time. (b) [±] The polynomial has a valid solution. (c) [±] Tractability can be controlled via the number of variables in the polynomial.	[±] Trade-off with strong usefulness.	[+] The property is proven by the authors.	(a) [-] The solution algorithm is deterministic, which favors centralization. (b) [+] See BTP 1c.
Proof of eXercise [109]	(a) [+] See BTP 7a. (b) [=] Unclear efficiency in practice.	(a) [+] <i>Generate</i> runs in constant time. (b) [=] Depends on the class of tasks. (c) [±] Miners are assigned sub-portions of the input matrix as tasks.	[±] Task hardness is reduced by dividing tasks into easier sub-problems.	[+] Valid solutions are always accepted.	(a) [±] Common solution algorithms are deterministic. (b) [±] See BTP 1c.
Proof of Search [108]	(a) [+] See BTP 7a. (b) [±] Solutions are verified in constant time.	(a) [+] <i>Generate</i> runs in constant time. (b) [±] A small task solution space may lead to unsolvable BTs. (c) [±] Tractability is guaranteed only when assuming a sufficient task solution space.	[±] It is not possible to reduce the difficulty of unsolvable instances.	[+] Valid solutions are always accepted.	(a) [=] Depends on the class of tasks and the solution search cost compared to cryptopuzzles. (b) [+] See BTP 1c.
Ofelimos [41]	(a) [+] See BTP 7a. (b) [=] Assumptions on SNARGs efficiency may not be satisfied in practice.	(a) [+] <i>Generate</i> in constant time. (b) [+] All tasks have a valid solution. (c) [±] Tractability follows from cryptopuzzles and DPLS.	[±] It may be possible to reduce the difficulty by adjusting T_2 under certain assumptions on DPLS.	[±] Trivial for cryptopuzzles. Derived from SNARGs for useful tasks.	(a) [=] Fairness claimed, but the proof is omitted. (b) [=] See BTP 1c.

(*Non-amortizability*) is not discussed in the paper and remains unclear whether solutions are amortizable, for example, by selecting a block for which the associated origin has an already known chain of primes. Similarly, BTP 2 (*Non-zero variability across BTs*) is not discussed in the paper, and due to the lack of analysis, it remains unclear to what extent the property is satisfied.

Probabilistic primality tests like Fermat’s are efficient, BTP 6b (*BT verification efficiency*), insure BTP 9 (*BT verification completeness*) and provide BTP 5 (*BT verification soundness*) with very high probability, especially when repeated multiple times.

BTP 10a (*No superlinear dependency on the resources*) is not discussed, and the extent of fulfilling it remains unclear.

From Dickson’s conjecture [35] and Schinzel’s hypothesis H [107], both widely believed to be true, it follows that for any positive number k there are infinitely many Cunningham chains of length k . Those conjectures guarantee the existence of a valid solution; thus, Primecoin satisfies both BTP 7b (*BT solvability*) and BTP 4a (*No reduction in solvability*). It remains unclear, however,

whether the expected time to find a valid solution to a new BT, BTP 4c (*No increase in solution time*), does not increase.

6.2 Attempts at replacing cryptopuzzles with threshold-based tasks

As mentioned above, earlier attempts at cryptopuzzle equivalence drew two lines of criticism on the proposed classes of tasks: their practical utility appeared insufficient and the stated goal of equivalence in terms of satisfying all cryptopuzzle properties was not achieved. A number of later attempts focused on utilizing classes of high practical utility (such as ML); they were mindful about the properties without expecting to reach the tantalizing ideal of full equivalence. Specifically, works in these category considered threshold-based decision tasks derived from useful optimization tasks. Valid solutions were those scoring above (or below) a predefined threshold according to some optimization metric; thus thresholds were used to emulate the difficulty of cryptopuzzles. The metric is typically related to usefulness, so suppliers are directly interested in solutions scoring better than the threshold. As extensively discussed in Section 5.1, the choice of threshold impacts BTPs at different levels and it is challenging to make a good decision regarding the threshold value.

The works in this group are based on ML [6, 24, 80] and other optimization tasks [50]. In the following, we discuss two works in ML.

DLchain [24] is a system that uses Deep Learning (DL) training as the class of tasks for a permissionless ledger. DL tasks are submitted by trusted suppliers that maintain a permissioned blockchain among themselves. The goal of this second ledger is to establish the order in which tasks need to be solved; however, DLchain does not elaborate further on the permissioned part. For example, it remains unclear how tasks and their relative order are disseminated to the solvers. Suppliers specify a *desired accuracy threshold* δ for every task, and DL models need to satisfy δ to be considered valid. If a task does not achieve δ within a system-wide deadline, the task is terminated and skipped. The protocol introduces intermediate milestones towards δ , called *short-term targets*. The idea is that the δ may take too long to achieve, so a block can be proposed as soon as the trained model satisfies an (easier) short-term target. At any rate, the mechanism's description is qualitative, and it remains unclear how reliable the short-term target mechanism is in providing timeliness guarantees. Unlike suppliers, solvers form a permissionless network and are free to join and leave the system. The solvers train the model on the data using a training seed S that depends on the block header. During training, solvers save training metadata into a list of off-chain checkpoints constituting the so-called *epoch chart* and include the epoch chart hash in the block header. As we explain below, the *epoch chart* is used as a reference point in the case of forks. Upon successful block inclusion, solvers are awarded the block reward and the transaction fees.

The proposal aims to achieve strong usefulness, and we note that all computing steps are spent on useful tasks. However, the system critically relies on two strong assumptions. The first assumption is that task suppliers pre-train a DL model to correctly estimate and assign the value of δ so that improving accuracy becomes challenging and stochastic. This creates a clear computational burden on the supplier side that may deter use. The second assumption is that improving the model performance up to δ is useful for the supplier. We argue that this may not necessarily be the case. Indeed, as already discussed in Section 5.4, several studies show that over-optimizing the DL model to the training data, as proposed in the solution, leads to loss of generality and may underperform with new data samples, undermining the motivation about usefulness.

In the proposed scheme, BTP 1a (*BT hardness*), BTP 7c (*BT tractability*) and BTP 7b (*BT solvability*) depend on the task and the short-term target values, both of which are unilaterally decided by the task supplier. Suppliers are assumed to be trusted; however, they may be genuinely too optimistic or pessimistic in setting δ and short-term targets. Those may be outright infeasible or intractable, impacting BTP 7b and BTP 7c respectively. For the same reason, BTP 4a (*No reduction in solvability*)

may not hold. However, DLchain counters that risk at the system level with the introduction of the system-wide deadline.

BTP 1b (*Context sensitivity*), on the other hand, is guaranteed due to using the training seed S . A model is considered valid only when it satisfies the short-term target. Verification is efficient in the lack of conflicts, which satisfies BTP 6b (*BT verification efficiency*). Upon conflict, however, nodes must use the *epoch chart* to verify step by step that the solution matches the chart, an extremely inefficient process. In such a situation, the winning branch is the one whose *epoch chart* has logged the biggest number of training iterations (epochs). Because of this overhead, BTP 6b is, in the worst case, as expensive as solving the task in the first place.

According to the authors, improving accuracy up to at least the value of a short-term target is a random trial and error process similar to the cryptopuzzle brute-force search. Therefore, the solution search is likely to guarantee a sufficient degree of randomness for BTP 2 (*Non-zero variability across BTs*), BTP 10a (*No superlinear dependency on the resources*), and BTP 4c (*No increase in solution time*). On the other hand, fairness is not explicitly considered by the authors as a property and deserves further validation.

Besides, the authors do not explicitly consider BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*). We argue that while tinkering with the short-term targets has an effect on the task difficulty, this approach is not as well-studied as the difficulty mechanism in PoW. Similarly, BTP 1c,10b (*Non-amortizability*) is not discussed, and the degree of non-amortizability remains unclear for DL tasks.

Both BTP 5 (*BT verification soundness*) and BTP 9 (*BT verification completeness*) hold because models with sufficient accuracy are accepted, and the network rejects models with insufficient accuracy. As a final remark, proposed blocks must include the model. However, the model size is non-negligible, and the space requirement will likely scale to several GB within the span of a fairly small number of appended blocks.

Coin.AI [6] replaces cryptopuzzles with the training of DNNs. Solvers can choose the training algorithm they prefer to implement *Solve*, and ML models are considered valid only if they perform above a certain quality threshold. The threshold decays over real-time, which ensures that a block is eventually proposed. While central to the design, Coin.AI does not explain how the quality threshold is initially set and how fast it decays, nor does it discuss the related synchronization challenges for geographically distributed nodes. *Generate* uses a surjective function to map the block hash value into an NN architecture and initial hyperparameters. This mapping function is implemented through formal context-free grammar. A modification to any block element (e.g. the block proposer) changes the input to the mapping and, consequently, the resulting BT.

Any node with sufficient funds to pay a task proposal fee can propose an instance to be solved. The instance is unaltered, so we classify the tasks under strong usefulness and note that all computing steps are spent on useful tasks. Conversely, solvers receive rewards for proposing blocks and including transactions in a way similar to Bitcoin. The authors discuss a few alternative options for the task selection policy, but details are not reported, and the description remains at a high level.

Both BTP 1a (*BT hardness*) and BTP 7c (*BT tractability*) rely on the hardness assumption of the training process. While it is commonly assumed that NNs are computationally hard to train, the time to produce a solution in practice depends on the interplay between the validity threshold, the NN, and the training data as discussed in Section 5.4. Coin.AI does not go into the details and properties of the context-free grammar and initial threshold selection, hence it is unclear whether BTP 1a and BTP 7c are satisfied. The context-free grammar approach used to generate BTs guarantees BTP 1b (*Context sensitivity*). On the other hand, it is unclear to what extent this approach prevents transfer learning, which is undesirable for BTP 1c,10b (*Non-amortizability*), as explained in 5.4. Indeed, classic transfer learning requires the layers of the pre-trained and the

receiving neural network to be compatible, an occurrence that is made unlikely by context-free grammar. On the other hand, we observe that solvers may rearrange block transactions to search for specific NN configurations. It, therefore, remains unclear whether BTP 1c,10b is satisfied.

Coin.AI implicitly assumes that all instances are “hard enough” and discusses the threshold decay mechanism, without analyzing the case in which the BT may be extremely easy and quickly solved by most NN. Thus, while BTP 8 (*Adjustable upper threshold*) is likely satisfied through the decay mechanism, we argue that BTP 3 (*Adjustable lower threshold*) is not.

Solvers must agree on the current time in order to calculate the current quality threshold correctly and guarantee both BTP 9 (*BT verification completeness*) and BTP 5 (*BT verification soundness*). However, it is not uncommon in blockchain systems to experience substantial time differences. The ramifications of loose synchronization and related incentives, while detrimental to the blockchain properties, are not discussed in Coin.AI. Verification is efficient, satisfying BTP 6b, since re-computing the mapping function and performing a single propagation of the dataset through the solution model are both fast operations.

BTP 2 (*Non-zero variability across BTs*) is not considered either. However, it is likely satisfied due to the heterogeneity of NN returned by *Generate*.

Difficulty decay ensures that even hard instances are solvable over time, thereby guaranteeing both BTP 7b (*BT solvability*) and BTP 4a (*No reduction in solvability*). On the other hand, BTP 4c (*No increase in solution time*) is not satisfied: upon change of context, the expected time to solve a new BT increases since miners have to abandon the currently trained NN and start anew. The chain selection rule is not explicitly described. The description hints that better-performing models weigh more to establish the blocks belonging to the main chain. In such a case, we observe that miners may prefer to keep solving the current BT rather than to switch to a new task, and therefore, it is unclear whether miners are incentivized to switch or rather to keep improving their model. BTP 10a (*No superlinear dependency on the resources*) is not discussed, however, as argued in Section 5.4, we believe that DL BTs do not satisfy BTP 10a. It remains to be seen in practice to what extent small solvers are able to propose blocks in Coin.AI.

6.3 Complementing useful tasks with a staking mechanism

One of the challenges in replacing cryptopuzzles with useful work is estimating and controlling the difficulty of useful tasks. The systems in this group [1, 21, 23, 63] complement solving useful tasks with an additional mechanism whose role is to contribute to controlling the overall difficulty. The exact mechanism differs across these systems but it invariably consists of PoS and staking elements, with an occasional addition of cryptopuzzles. The staking mechanisms indeed contribute to controlling the difficulty: supplier-side deposits may help with BTP 7b and BTP 7c since it becomes the supplier’s interest that the task is solvable and tractable. Similarly, solver-side deposits may be used when committing to a solution, hence deterring task reuse and helping with BTP 1b. It should be noted, however, that the security of collateral payment systems is, unlike Bitcoin, based on the assumption of rational attackers.

Hybrid Mining (HM) [21] proposes a hybrid PoW system in which solvers choose whether to solve NP-complete tasks or cryptopuzzles. Block proposers are awarded the transaction fees and either a task solution reward or the block reward, depending on the task type included in the block. The task solution reward is paid by the supplier instead of being newly minted, as happens with the block reward. Thus, the economics of HM dictate that at least half of the blocks are secured by cryptopuzzles. Hence, the fraction of useful work performed by the system is measured in blocks rather than computing steps. HM does not explicitly specify whether the useful tasks must be decision or optimization problems. Boolean satisfiability problems (SAT) are used throughout the paper as the working example. SAT is a decision problem, and therefore, in the following analysis,

we assume this flavor of tasks. We note that if no variable assignment exists such that the Boolean formula evaluates to true, the SAT instance is unsatisfiable, incurring all the challenges related to verifying negative claims, as discussed in Section 5.1.

Tasks are advertised as transactions over the network. Suppliers lock several funds in the advertisement, including the task solution reward and an advertisement fee equivalent to the block reward, to deter collusion between solvers and suppliers. Any node can submit a task regardless of complexity as long as all fees and deposits are paid. Usefulness is not clearly defined, but instead informally mentioned as *problems of immediate practical value to their owners*. The proposed system does not modify the useful task based on the context so that it satisfies strong usefulness based on Definition 4. To prevent stealing task solutions, HM uses a two-phase mechanism to safely disclose solutions. In the first phase, miners propose a block containing a claim that a specific task was solved. Such a block does not contain any PoW but rather a security deposit. The task solution is thereafter revealed as a blockchain transaction. If a solver fails to disclose the useful task solution within a predefined number of blocks, the solver loses the associated security deposit. At any rate, a useful block that has been included in the chain remains part of the chain regardless of correct solution disclosure. HM retains Bitcoin’s longest chain selection rule. However, the authors admit that the difficulty of problem instances is not well-defined and is hard to compare across tasks. Therefore, the main chain in HM is decided based only on which of the competing branches has the most cryptopuzzle-secured blocks. Since solvers are free to choose whether to solve cryptopuzzles or useful tasks, we argue that they are incentivized to solve cryptopuzzles to increase their chances of block inclusion. In the following, the analysis refers mainly to the useful task construction since cryptopuzzle properties have been extensively described in Section 3 and can be easily inferred. Unless otherwise noted, a property is only satisfied at the system level if both useful tasks and cryptopuzzles satisfy it.

In the proposed construction, there is no guarantee on the BTP 1a (*BT hardness*) of a task: suppliers may provide a series of very simple instances, or malicious solvers may have enough funds to disseminate non-solved useful blocks, causing an undue increase of the BP rate. The latter case, however, would deplete the funds of the malicious miners. On the other hand, submitted tasks may not have a correct solution (a valid Boolean assignment to the SAT clauses that evaluates to true), jeopardizing BTP 7b (*BT solvability*) and indirectly BTP 7c (*BT tractability*). BTP 2 (*Non-zero variability across BTs*) as well as BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*) are not discussed, and while they hold for cryptopuzzles, it is unclear how arbitrary computational tasks may fare with respect to those properties. BTP 1b (*Context sensitivity*) is not satisfied by useful tasks since the solution to the useful task does not depend on the context and can be solved irrespective of the state of the chain.

BTP 4a (*No reduction in solvability*) is not considered in the work and depends on the class of tasks. For example, the probability of existence of a valid solution for the new BT is reduced if the SAT instance is unsatisfiable. On the other hand, miners may opt to solve cryptopuzzles which do not reduce such probability, however, at the expense of usefulness. An interesting observation is that, since useful blocks do not count for the longest chain rule, the incentive for miners to switch to a newly received useful block is unclear since ignoring it yields no penalty, and extending it yields no benefit.

BTP 4c (*No increase in solution time*) must be analyzed separately for cryptopuzzles and SAT instances. The expected time to find a valid solution for a cryptopuzzle does not increase; on the other hand, even assuming the existence of a valid solution for a given SAT instance, the time to find a solution may be longer. Thus, BTP 4c depends on the class of tasks, and it is not satisfied in the case of SAT.

Some properties are not considered in HM. Specifically BTP 1c,10b (*Non-amortizability*), BTP 6b (*BT verification efficiency*), BTP 5 (*BT verification soundness*), BTP 9 (*BT verification completeness*). BTP 6b, BTP 5, BTP 9 hold for SAT problems; however, those properties may not be guaranteed for other classes of tasks. Regarding BTP 10a (*No superlinear dependency on the resources*), useful tasks such as SAT instances may lead to a superlinear probability increase. We conjecture the lack of fairness as a likely problem and raise this point as an open question requiring further analysis.

6.4 Domain transformation

The group of Domain transformation uses techniques that transform the computation of the useful task from one domain to another to improve the BTP coverage. The solution can be quickly and verifiably reconstructed (see Section 4.5) from the miners' response. This approach requires strong theoretical foundations and applies to very specific classes of tasks. Proofs of Useful Work [8] belongs to this group.

Proofs of Useful Work (PoUW) proposes a system that replaces cryptopuzzles with the evaluation of low-degree polynomials derived from instances of k-OV. Bigger k-OV instances generate polynomials with more terms, which are harder to solve. An updated version [9] of PoUW retracts the definition of usefulness and omits the design of the blockchain system; in the following, we analyze the original PoUW paper, including the system design, because we believe it is an important example worth deeper analysis. The original usefulness definition is "Computational tasks can be delegated as challenges to the workers such that the solution to the delegated task can be quickly and verifiably reconstructed from the workers' response" [8]. No external interest is mentioned in this definition; however, we classify PoUW as satisfying strong usefulness since a *Solve* output contains the result of a specific user-supplied instance. All computing steps are spent on useful tasks. PoUW does not define a supply model and assumes unlimited tasks collected in a publicly accessible board. We believe this to be a strong assumption since the progress of the chain relies on the availability of those tasks. At any rate, in the following analysis, we assume that tasks are always available. Miners select arbitrarily which task to solve among the ones on the board.

It remains unclear what incentivizes solvers to select a bigger OV instance, thus requiring more time to solve the corresponding polynomial. In general, the incentives of the construction are not described. The k-OV instance, along with a block-dependent nonce, is used to *Generate* a polynomial whose evaluation is proven to be hard in the average case. For a tutorial on average-case fine-grained complexity theory, refer to [7].

Unlike the simple design in Section 5.2, PoUW tasks are not restrained to $k = 2$. Tasks may have higher polynomial complexity, and the authors go to lengths to prove that the low-degree polynomials are hard in the average case. However, we deem BTP 1a (*BT hardness*) as likely satisfied (as opposed to fully satisfied) because no observation is made about the storage requirements of the matrices which, as discussed in Section 5.2, can be substantial. In addition, the authors prove BTP 1c,10b (*Non-amortizability*) for polynomials, and the block-dependent nonce in the construction supports BTP 1b (*Context sensitivity*). On the other hand, BTP 2 (*Non-zero variability across BTs*) is unlikely to be satisfied: *Solve* is deterministic, and miners with similar computing power are likely to propose a block at roughly the same time. BTP 7b (*BT solvability*) and BTP 4a (*No reduction in solvability*) are satisfied since all k-OV tasks have a solution. BTP 7c (*BT tractability*) is not discussed, but we surmise that BTP 7c can be satisfied by controlling the number of variables in the polynomial.

Similarly, BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*) are not discussed either. Since the size and degree of the polynomial are derived from the k-OV instance, a given instance determines the hardness of the task. Based on our understanding, the derivation

Table 7. Usefulness and supply models, task selection policies and incentives of the selected proposals.

Paper	Usefulness Model	Supply Model	Task Selection Policy	Incentives
Primecoin [69]	Weak usefulness All computing steps are spent on useful tasks.	Miners generate BT independently like in Bitcoin.	Not applicable	Not discussed
DLchain [24]	Strong usefulness All computing steps are spent on useful tasks.	Restricted to suppliers within a permissionless blockchain	Restricted to a single task at a time decided by the consortium of suppliers	Block reward and transaction fees
Coin.AI [6]	Strong usefulness All computing steps are spent on useful tasks.	Any currency holder in the network can submit a task.	Possible Task Selection Policies are discussed at a high level.	Block reward and transaction fees
Hybrid Mining [21]	Strong usefulness At most half of the appended blocks contain useful tasks.	Any node in the network can submit a task.	Solvers decide which tasks to solve, either a cryptopuzzle or a useful task.	Proposer receive transaction fees and either solution reward or coinbase reward.
Proofs of Useful Work [8]	Strong usefulness All computing steps are spent on useful tasks.	Not discussed	Arbitrary task selection from a public board	Not discussed
Proof of eXercise [109]	Strong usefulness All computing steps are spent on useful tasks.	Any node in the network can submit a task.	Solvers are assigned randomly the task to solve.	Deposits to disincentivize misbehaving and block reward to incentivize task solutions.
Proof of Search [108]	Strong usefulness Unknown, highly dependent on the solution search algorithm	Any node in the network can submit a task.	Restricted to a list of tasks embedded in the last valid block.	Separate rewards for proposing a block and for the best solution to the useful task.
Ofelimos [41]	Strong usefulness At most half of the computing steps are spent on useful tasks.	Not discussed	Tasks are assigned to a number of consecutive blocks. Task prioritization is based on an unspecified scheduling algorithm.	Rewards are given to ranking blocks and input-blocks proposers. Transaction fee assignment is similar to Bitcoin.

mechanism does not naturally accommodate hardness adjustment mechanisms, and hence, there is a trade-off with strong usefulness.

BTP 4c (*No increase in solution time*) is not satisfied because a miner switching tasks “loses” the computation done so far, with the impact described in Section 4.4.1.

The authors prove BTP 5 (*BT verification soundness*) and BTP 9 (*BT verification completeness*) of their scheme. BTP 6b (*BT verification efficiency*) is shown to have linear complexity $\tilde{O}(n)$, as a function of the number of vectors in the k-OV instance. On the other hand, in the general case, the low-degree polynomial may include many terms whose number is proportional to the k-OV instance size, leading to proofs of non-negligible size. Therefore, storage requirements may be an important practical constraint in a real deployment. BTP 10a (*No superlinear dependency on the resources*) is not explicitly considered in PoUW, and the solution search algorithm is deterministic: the probability of proposing a block increases superlinearly with the relative amount of resources spent. This raises concerns about the decentralization of the solution.

6.5 Shuffling of tasks

Altering a supplied task based on the context raises the issue of strong usefulness described in Section 4.4. Therefore, a group of works enforces context sensitivity through the process that assigns the task to a miner rather than by modifying the task itself. This process is random but verifiable and uses an assignment service that has been referred to as shuffling. Proof of eXercise [109] is the first to propose that approach, followed by [114]. We analyze the former in depth.

Proof of eXercise (PoX) is a system in which cryptopuzzles are replaced by matrix-based computation problems, motivated by the claim that such problems span a wide range of useful real-world use cases and that they are an important abstraction for many scientific computation tasks. PoX claims to support a broad set of matrix operators, including sum, product, Schur product,

etc. In the proposed system, two shuffling services match tasks and nodes in a uniformly random fashion. The first shuffling service assigns miners to the task to be solved, while the second service assigns the output of the *Solve* function to a subset of validators. The former service is used to limit the effectiveness of collusion attacks, while the latter is to limit the bias in verification and to improve scalability. We note, however, that while being central to the PoX design, shuffling services are underspecified. PoX suggests implementing such services on TOR and onion routing but does not include the discussion of challenges, practicality, and incentives.

The paper does not provide a clear definition of usefulness. However, the proposed solution seems to target strong usefulness as in Definition 4. All computing steps are spent on matrix operations. Any node can supply a task by storing the task instance in a highly accessible database and thereafter advertising the task in a blockchain transaction. While task suppliers and solvers pay a deposit that is returned if they do not misbehave, block proposers are incentivized to solve the task to collect the coinbase reward.

The proposed design assumes the existence of a routine to efficiently assess the task runtime based on the matrix sparseness and algebraic structure; such estimate is attached to the task as a *Proof of Hardness* (PoH). To ensure BTP 1a (*BT hardness*), accepted tasks are those whose PoH exceeds a pre-defined threshold. The mechanism to satisfy BTP 1b (*Context sensitivity*) is based on the pseudo-random task selection: if the block content is modified, the pseudo-random matching will very likely be invalidated. Therefore, the security of such an approach crucially relies on (1) the implementation of the shuffling service and (2) the sets of supplied tasks and solvers being large enough. The author suggests matching miners and tasks through the hash of the block header. We observe that such matching requires active miners to be registered on the blockchain. Besides, such membership must be maintained amidst node churn. BTP 1c,10b (*Non-amortizability*) is not discussed in the work; we argue that it is possible to amortize the solution search across instances if there is substantial overlap among them. However, the practicality of such an attempt depends on the specific class of tasks.

Similarly, BTP 7b (*BT solvability*) and BTP 4a (*No reduction in solvability*) depend on the type of matrix operation. Matrix multiplication admits a solution as long as the dimensions of the input matrices are compatible. However, if matrices represent a system of linear equations that cannot be satisfied simultaneously, the task has no solution [111], thus invalidating BTP 7b and possibly BTP 4a. Because of this ambivalence, both BTP 7b and BTP 4a depend on the specific class of tasks. A similar consideration applies to BTP 7c (*BT tractability*). Assuming BTP 7b is satisfied, BTP 7c can be obtained in many matrix-based computations by decomposing the task into sub-problems.

The proposal does not discuss BTP 2 (*Non-zero variability across BTs*). While the shuffling service may help, it remains unclear in practice whether BTP 2 is satisfied. The paper suggests that BTP 3 (*Adjustable lower threshold*) may be attained by batching submitted tasks together. On the other hand, difficulty reduction BTP 8 (*Adjustable upper threshold*) can be achieved by assigning sub-problems to miners, as argued for BTP 7c. In the case of a new block reception, however, the change of context invalidates the current task, and the average time to find a solution to the new task is higher. This is detrimental to BTP 4c (*No increase in solution time*), and thus miners may not be incentivized to accept incoming blocks. The proposed system includes the role of auditors, which validate proposals. BTP 5 (*BT verification soundness*) is therefore satisfied as invalid proposals are likely to be rejected in proportion to the number of verifications performed by the auditors.

The system includes a probabilistic multi-step verification algorithm involving the shuffling service and multiple auditors, with each auditor concurrently verifying a portion of the solution. While dividing the verification among multiple auditors speeds up the process, efficiency in practice remains an open question. On the other hand, the verification algorithm satisfies BTP 9 (*BT verification completeness*), as valid solutions are never rejected.

BTP 10a (*No superlinear dependency on the resources*) is not considered in the work. In practice, the impact of shuffling on fairness remains unclear. We observe that common solution algorithms for matrix operations are deterministic, thus favoring the centralization of the computation. A drawback of matrix-based tasks is the large input size that is necessary for computation hardness, affecting both storage and bandwidth requirements.

6.6 Solution search while mining

In this group, proposals retain the use of cryptopuzzles while expanding the block proposal process to incorporate the exploration of solutions for useful tasks. For example, candidate cryptopuzzle nonces can be derived from valid solutions to the useful tasks or a solution to the cryptopuzzle is used as a seed for solving the useful task. In both cases, miners are motivated to explore a vast solution space of the useful task. Proof of Search [108], Ofelimos [41], Axechain [127], Proof-of-Evolution [15], mPoW [3], as well as the system of [77] belong to this category.

Proof of Search (PoS) is a system where solvers explore the solution space of useful optimization problems while solving cryptopuzzles to propose blocks. First, solvers prepare a block template containing transactions, etc. Then, solvers start testing different nonces so that the output of the hash function is below an established difficulty threshold, similar to Bitcoin cryptopuzzles. Unlike Bitcoin, however, PoS nonces are not random sequences of bits; instead, they must include a solution (not necessarily good wrt. the optimization metric) to an optimization problem. Additionally, nonces include a context-sensitive string derived from the solution and from the block context. The work proposes a complicated scheme to this end, which consists of computing an optimization metric for the solution and skewing this metric based on the block context. However, we believe that the same goal can be achieved in a simpler way and with stronger guarantees by applying a *contextualizer* function. All solvers and validators must agree upon the contextualizer function; it can, e.g., be provided along with the task. The simplest implementation of the contextualizer function would be a secure hash.

Due to the hardness of cryptopuzzles, many different solutions are tested while proposing a block, increasing the likelihood of finding a good solution with respect to the optimization metric. To prevent a solution from being stolen by other nodes, PoS uses a mechanism akin to a commitment scheme where every node first registers on the chain the best value found. The way it is implemented is not specified in the paper, however. After a few blocks, each node checks whether its solution is the best, and if so, it discloses the solution. Thus, suppliers receive the best solution found by the whole network.

Unfortunately, the conflict resolution rule is not presented. At the same time, the work provides an interesting discussion of the incentive model. In particular, the system assigns a reward for proposing a block and a separate reward, paid by the supplier, for the best solution disclosed. The work explains why the distribution of rewards disincentivizes suppliers and solvers to collude. Suppliers submit tasks as transactions that are included in the blockchain, and any node can be a supplier. However, how miners select which tasks to solve is not discussed. PoS does not define usefulness, yet we classify it as strong usefulness since the proposed solution complies with Definition 4. Miners dedicate a portion of their computational efforts to finding a useful task solution. The proportion of computation spent on useful tasks as opposed to cryptopuzzles highly depends on the solution search algorithm (which is solver is free to choose) as well as on the relative difficulty of the optimization problem, making it difficult to generalize. Nevertheless, we note that miners may prefer cheaper algorithms, such as random permutations of TSP cities, over better quality producing but slower and more expensive algorithms. This dichotomy may lead to the generation of lower-quality solutions overall.

Most BTPs are trivially satisfied by the use of cryptopuzzles; however, we have identified a few that are affected by the introduced nonce scheme. Among the satisfied properties, BTP 1a (*BT hardness*) and BTP 2 (*Non-zero variability across BTs*) follow naturally. BTP 1b (*Context sensitivity*) leverages the contextualizer function and is supported like in Bitcoin. Similarly, BTP 1c,10b (*Non-amortizability*) follows from Bitcoin.

The verification leaves no probability of invalid proposals being accepted, thus supporting BTP 5 (*BT verification soundness*). Besides, the proposed verification algorithm is both efficient and complete, satisfying both BTP 6b (*BT verification efficiency*) and BTP 9 (*BT verification completeness*). By assuming that finding a solution to the useful task is cheap compared to cryptopuzzles, BTP 10a (*No superlinear dependency on the resources*) derives from cryptopuzzles. In practice, such cost depends on several factors, such as the choice of useful tasks and the available solution algorithms; it remains unclear whether PoS satisfies BTP 10a.

An unexpressed assumption of PoS is that the solution space of the useful task instance is large enough to include a valid cryptopuzzle solution. Imagine the trivial example of a TSP of three cities: there exist only six possible arrangements, hence no more than six nonces to test. We observe, therefore, that an instance with a small solution space may have a very low probability to yield any valid block. This potential issue is exacerbated by the fact that suppliers may be malicious and supply small instances on purpose. This is a challenge for BTP 7b (*BT solvability*) and BTP 7c (*BT tractability*), with the consequences described in Section 3.2. The main difference from cryptopuzzles in Bitcoin is that the size of the nonce space in Bitcoin is fixed and controlled by the system while the solution space of the useful task instance in PoS is outside the system control. The size of the solution space affects BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*) asymmetrically. BTP 3 is guaranteed regardless, since it is always possible to arbitrarily increase the difficulty of the cryptopuzzle. However, BTP 8 is not satisfied. In particular, this is true for cases where there is no solution because it is not possible to reduce the difficulty of unsolvable instances. Sudden variations in the size of the solution space across different instances may also have an unpredictable effect on BTP 4a (*No reduction in solvability*) and BTP 4c (*No increase in solution time*). Besides, the committing scheme to disclose the solutions is likely to face serious scalability issues due to the high number of tasks and nodes.

Ofelimos is a blockchain protocol whose consensus is built around a stochastic local search algorithm called Doubly Parallel Local Search (DPLS). Stochastic local search is a generic algorithmic paradigm for solving optimization problems, and DPLS is designed to be widely applicable to many classes of optimization tasks. The core of the mining algorithm consists of executing the DPLS, which constitutes the useful part of the PoW. However, to defend against pre-computation attacks and cherry-picking instances of low complexity, miners must first solve a cryptopuzzle of moderate difficulty T_1 . The resulting nonce value is used as a seed for the DPLS algorithm, thereby randomizing the computation. Miners feed the DPLS output into one single post-hashing round that decides, against an adjustable threshold T_2 , whether the block is eligible for publication. Note that the miner only learns whether a PoW attempt is successful after executing DPLS, i.e., the computation cannot be cut short to speed up block creation. A miner repeats the computation sequence of solving a cryptopuzzle, performing useful work, and calculating one post-hash, until the post-hash of a sequence lies below T_2 , allowing for the block to be published. The authors estimate that in the common case, less than half of the total computation is spent on the useful task, while the rest is dedicated to cryptopuzzles. However, they note that it is possible, in restricted cases related to DPLS hardness, to increase this fraction arbitrarily close to one.

A block contains two points explored using the DPLS algorithm: the one that leads to a sufficiently small post-hash and enables block creation, and the best one in terms of the optimization metric. The latter point is used to progress the DPLS algorithm.

Ofelimos incorporates two additional optimizations: (a) the cost of block verification is reduced by having miners append a Succinct Non-interactive ARGument (SNARG) proving the correctness of both exploration points contributing to the block, and (b) a 2-for-1 scheme [43, 96] is employed to publish local search results faster, without waiting until the block is proposed. The useful tasks to be solved are selected locally based on public information posted on the blockchain; however, technical details are not presented in the work. The supply model is not described either. Nevertheless, we classify usefulness as strong since task instances are not modified.

Many BTPs stem from the cryptopuzzle-based pipeline. BTP 1a (*BT hardness*) and BTP 7c (*BT tractability*) follow from the individual hardness of both cryptopuzzles and DPLS, though the work does not provide an exact analysis of the combined hardness. Tasks are solvable, satisfying BTP 7b (*BT solvability*), because miners in Ofelimos can always start the cycle of cryptopuzzle, DPLS, and cryptopuzzle again so that eventually, a solution would be found. Because of the solvability, BTP 4a (*No reduction in solvability*) holds as well. BTP 4c (*No increase in solution time*) is satisfied when the same values of T_2 are consistently used, resulting in the same average time to find a solution. Increasing or decreasing the value of T_2 may help with supporting BTP 3 (*Adjustable lower threshold*) and BTP 8 (*Adjustable upper threshold*), respectively. However, the exact effect of a given change in T_2 on the difficulty of the entire BT, which consists in the three-step pipeline, is difficult to calculate or predict.

BTP 2 (*Non-zero variability across BTs*) follows due to the stochastic nature of DPLS and the well-studied variability of cryptopuzzles. Finally, the pre-hashing step ensures the context-dependency of BTP 1b (*Context sensitivity*).

BTP 1c,10b (*Non-amortizability*) is guaranteed for the hashing part, and it is likely satisfied for DPLS because of the pre-hashing step, which prevents miners from cherry-picking useful computations.

The protocol requires the computation of a SNARG to secure DPLS solutions and verifies the result in polynomial time. To this end, Ofelimos assumes a bounded number of computation steps to produce and verify the SNARG. No discussion is provided on how realistic such an assumption is given the available SNARG implementations, and thus the impact on BTP 6b (*BT verification efficiency*) remains unclear. BTP 5 (*BT verification soundness*) and BTP 9 (*BT verification completeness*) are not discussed, but we conclude they are both satisfied due to the properties of SNARGs [48] and cryptopuzzles. Finally, the authors claim that any set of solvers will produce a subset of blocks roughly proportional to their computing power, resembling BTP 10a (*No superlinear dependency on the resources*). However, details and proofs are omitted from the work.

7 RESEARCH GAPS AND OUTLOOK TO THE FUTURE

In the following, we point out selected research directions and gaps.

Formalization: This work provides extensive, albeit somewhat informal, reasoning for the importance of cryptopuzzle properties. An extension to the current work may formalize the BPPs as this paper did with BTPs, as well as include formal proofs of sufficiency and necessity. Furthermore, the analysis summarized in Tables 5 and 6 shows that none of the proposed systems satisfies all the identified properties, corroborating the perceived difficulty of the Proof-of-Useful-Work problem. We conjecture that the classes of tasks, such as cryptopuzzles, that satisfy all of the properties are those whose most efficient solution algorithm is a brute-force search. In other words, the solution search algorithm can be modeled as a perfect random number generator. However, this conjecture remains unproven for the time being.

Characterization of property support: It is a challenge to draw definitive conclusions on the BTP support given a specific class of tasks. In fact, unlike cryptopuzzles, such analysis must account for the heterogeneity of solution algorithms and problem instances. While NP-complete problems

are provably hard, there exists no equivalence class in terms of variability (BTP 2) or fairness (BTP 10), to name a few. The BTPs we identified in this work can be evaluated experimentally only by introducing assumptions and restrictions on the solution algorithms and instance sets. In addition, those types of evaluations are not common in the current research literature, which rather focuses on other types of metrics. For example, heuristics for the TSP show extensive results on the solution quality [46, 103] as well as the runtime [11] without, generally, analyzing variability. ML literature elaborates extensively on the model accuracy [12, 105] and convergence [125] while providing little insight on, for example, how the performance varies with the allocated computing power. Therefore, we believe there is room for significant new research to investigate less explored metrics and characterize the property support of state-of-the-art algorithms.

Reliable estimation and difficulty adjustment mechanisms: While NP-complete problems are provably hard, not all instances are hard to solve in practice. One of the features of cryptopuzzles essential in the context of PoW is that, given a fixed difficulty, all cryptopuzzle instances require, on average, the same solution time. Consequently, solution time is reliably adjusted by changing difficulty. The concept of difficulty in useful tasks, however, is not sufficiently defined. What parameter should be adjusted to increase the solution time e.g. twofold? How does this affect all instances? Systems such as [109] invoke the support of a practical mechanism to estimate the solution time. We note that the soundness and availability of reliable and practical mechanisms to estimate the average solution time would greatly strengthen proof of useful work proposals. The benefit of having reliable estimates is multifold: such estimates can drive task selection policies to improve, for example, BTP 10. They can also help set the best validity threshold, which is a key issue for many important classes of tasks, as described in Section 5.1.

Models: We observe insufficient exploration of (1) comprehensive supplier models and task selection policies and (2) economic models and incentive mechanisms. As extensively discussed, classes of tasks vary broadly wrt. property coverage. However, the analysis of such coverage requires clear models and policies. Nodes may be honest, malicious, or rational. Task supply can be external or internal; task selection coordinated or uncoordinated. None of the proposals surveyed in Sections 6 and 8 comprehensively cover the possible supplier models and task selection policies, or discuss the trade-offs between various possibilities. While this work has taken the first step toward (1), there is still much work left to be done. We believe that such an analysis is necessary to complement the landscape of design choices.

8 RELATED WORK

The term *proof-of-work* preceded blockchains and was first formalized by M. Jakobsson and A. Juels in 1999 [64]. In the PoW scheme, a prover demonstrates to a verifier that a certain amount of computational effort has been expended in a specified time interval. Among the proposed PoW schemes [38, 120], hashcash [5] is widely believed to have inspired Bitcoin’s cryptopuzzles. Since then, repurposing Bitcoin’s cryptopuzzles for useful tasks has been an open problem. However, comparatively little effort has been directed into clearly defining cryptopuzzle properties and analyzing the challenges involved in its replacement.

Tentative property analysis has been conducted in [22, 37, 84, 106], while a more systematic, but still partial, analysis is presented in the book by Narayanan et al. [89]. At the time of writing, the most complete list of properties has been proposed by Ball et al. [8], which also provides a task interface that inspired the interface defined in Section 2. Nonetheless, a few important properties are not considered in [8], most notably properties about decentralization and fairness, as well as switchability. In summary, to the present day, there exists no commonly accepted list of properties that blockchain’s classes of tasks need to satisfy. To the best of our knowledge, the present work

includes the most complete definitions of properties, and it is the first to critically discuss their interplay in a systematic and extensive way.

The property framework developed in the current work is used to critically assess proposed systems in the literature. Indeed, there are many proposed systems and candidates for cryptopuzzle replacement. Useful tasks include, but are not limited to, prime number operations [42, 69], OV and linear algebra calculations [8, 109], Monte Carlo algorithms [32, 55], storage-intensive searches [86], DNA sequence alignment [60], and generic tasks [126, 127]. The most popular class of tasks in the proposed works are optimization tasks [23, 41, 63, 108, 114] sometimes restricted to the special case of ML, DL and RL [6, 18, 24, 25, 73, 74, 77, 80, 85, 87, 99, 118], Federate Learning [79, 100] and Genetic Algorithms [3, 15]. Due to the required hardness, some systems employ NP-complete problems [20, 21, 50, 93], among which TSP is the most frequently chosen class [75, 82, 112]. Finally, some interesting works propose a novel difficulty adjustment mechanism [97] and a novel reward system [71]. As explained in Section 6, some of the works [1, 18, 60, 63, 79, 86, 87, 126] are excluded from our analysis in Section 6 because they are using different trust or synchronization models compared to Bitcoin.

9 CONCLUSION

The possibility of using the extensive computing power supplied by blockchain miners is tantalizing and motivates the development of our property framework. This paper identifies and defines several important blockchain properties. We discuss the relevance of each property and the relation between them, highlighting trade-offs and challenges when replacing cryptopuzzles with a different class of tasks. The property framework has been used to assess the pitfalls and challenges of several blockchain designs that replace cryptopuzzles with useful work. In several cases, the proposed designs address a subset of our properties. Our analysis reveals that usefulness is in direct conflict with many important properties. Moreover, no alternative task that can satisfy all properties and completely replace cryptopuzzles has been proposed. With our framework, we hope to provide standard requirements for useful blockchain developers and researchers and to spark additional research efforts in this direction.

ACKNOWLEDGMENTS

We are thankful to many researchers who proofread an earlier version of this survey and provided insightful comments. A complete list of acknowledgments will be provided in subsequent versions.

REFERENCES

- [1] D. Amar and L. Zilpa. 2019. Incentive-Based Ledger Protocols for Solving Machine Learning Tasks and Optimization Problems via Competitions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*.
- [2] Derek Armstrong and Sheldon Jacobson. 2003. Studying the Complexity of Global Verification for NP-Hard Discrete Optimization Problems. *Journal of Global Optimization* 27 (2003).
- [3] Takaki Asanuma and Takanori Isobe. 2023. mPoW: How to Make Proof of Work Meaningful. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 106, 3 (2023), 333–340.
- [4] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts SoK. In *International Conference on Principles of Security and Trust*.
- [5] Adam Back. 2002. *Hashcash - A Denial of Service Counter-Measure*. <http://www.hashcash.org/papers/hashcash.pdf>
- [6] Alejandro Baldominos and Yago Saez. 2019. Coin.AI: A Proof-of-Useful-Work Scheme for Blockchain-Based Distributed Deep Learning. *Entropy* 21, 8 (2019).
- [7] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. 2017. Average-case fine-grained hardness. In *ACM SIGACT Symposium on Theory of Computing*.
- [8] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. 2017. Proofs of Useful Work. *Cryptol. ePrint Arch.*, Paper 2017/203.

- [9] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. 2018. Proofs of work from worst-case assumptions. In *Advances in Cryptology (CRYPTO)*.
- [10] Sergio Barrachina, Adrián Castelló, Mar Catalán, Manuel F. Dolz, and Jose I. Mestre. 2021. Using machine learning to model the training scalability of convolutional neural networks on clusters of GPUs. *Computing* 105, 5 (2021).
- [11] Jon Louis Bentley. 1990. Experiments on traveling salesman heuristics. In *Symposium on discrete algorithms*.
- [12] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag.
- [13] George Bissias and Brian Neil Levine. 2020. Bobtail: Improved Blockchain Security with Low-Variance Mining. In *NDSS*.
- [14] bitmain.com. 2022. *Bitmain Antminer*. <https://shop.bitmain.com/>
- [15] Francesco Bizzaro, Mauro Conti, and Maria Silvia Pini. 2020. Proof of Evolution: leveraging blockchain mining for a cooperative execution of Genetic Algorithms. In *IEEE International Conference on Blockchain*.
- [16] boinc.berkeley.edu. 2014. *BOINC*. <https://boinc.berkeley.edu/trac/wiki/DesktopGrid>
- [17] Colin Boyd and Christopher Carr. 2018. Valuable Puzzles for Proofs-of-Work. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*.
- [18] Felipe Bravo-Marquez, Steve Reeves, and Martín Ugarte. 2019. Proof-of-Learning: A Blockchain Consensus Mechanism Based on Machine Learning Competitions. In *Int. Conference on Decentralized Applications and Infrastructures*. 119–124.
- [19] Cambridge Centre for Alternative Finance. 2022. *Cambridge Bitcoin Electricity Consumption Index*. Retrieved October, 2022 from cbeci.org/cbeci/comparisons
- [20] Diptendu Chatterjee, Prabal Banerjee, and Subhra Mazumdar. 2023. Chrisimos: A useful Proof-of-Work for finding Minimal Dominating Set of a graph. [arXiv:2308.04407](https://arxiv.org/abs/2308.04407) [cs.CR]
- [21] Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Arash Pourdamghani. 2019. Hybrid mining: exploiting blockchain’s computational power for distributed problem solving. In *ACM/SIGAPP Symposium on Applied Computing*.
- [22] Yash Chaurasia, Visvesh Subramanian, and Sujit Gujar. 2021. PUPoW: A Framework for Designing Blockchains with Practically-Useful-Proof-of-Work & VanityCoin. In *IEEE International Conference on Blockchain*.
- [23] Canhui Chen, Zerui Cheng, Shutong Qu, and Zhixuan Fang. 2023. Crowdsourcing Work as Mining: A Decentralized Computation and Storage Paradigm. In *Asia-Pacific Workshop on Networking*.
- [24] Changhao Chenli, Boyang Li, and Taeho Jung. 2020. DLchain: Blockchain with Deep Learning as Proof-of-Useful-Work. In *SERVICES*.
- [25] Changhao Chenli, Boyang Li, Yiyu Shi, and Taeho Jung. 2019. Energy-recycling Blockchain with Proof-of-Deep-Learning. In *IEEE International Conference on Blockchain and Cryptocurrency*.
- [26] Nicos Christofides. 2022. Worst-case analysis of a new heuristic for the travelling salesman problem. In *Operations Research Forum*.
- [27] CoinMarketCap. 2023. *Today’s Cryptocurrency Prices by Market Cap*. <https://coinmarketcap.com/>
- [28] ISO/TC 211 Committee. [n.d.]. *ISO/TC 211*. Retrieved March 15, 2024 from <https://www.iso211.org/>
- [29] William Cook. 2020. *Concorde TSP Solver*. <http://www.math.uwaterloo.ca/tsp/concorde.html>
- [30] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). The MIT Press.
- [31] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2015. Geppetto: Versatile Verifiable Computation. In *IEEE Symposium on Security and Privacy*.
- [32] Alex Coventry. 2012. NooShare: A decentralized ledger of shared computational resources. *Rep., Apr* (2012).
- [33] Lawrence Davis. 1987. Genetic algorithms and simulated annealing. (1987).
- [34] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the Bitcoin network. In *IEEE P2P*.
- [35] Leonard E Dickson. 1904. A new extension of Dirichlet’s theorem on prime numbers. *Messenger of Math* 33 (1904), 155–161.
- [36] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. 2021. Survey on Blockchain Networking: Context, State-of-the-Art, Challenges. *ACM Comput. Surv.* 54, 5 (2021).
- [37] Maya Dotan and Saar Tochner. 2021. Proofs of Useless Work – Positive and Negative Results for Wasteless Mining Systems. [arXiv:2007.01046](https://arxiv.org/abs/2007.01046) [cs.CR]
- [38] Cynthia Dwork and Moni Naor. 1993. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology (CRYPTO)*.
- [39] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol. In *USENIX Symposium on Networked Systems Design and Implementation*.
- [40] Uriel Feige, Amos Fiat, and Adi Shamir. 1988. Zero-knowledge proofs of identity. *J. of Cryptology* 1, 2 (1988), 77–94.
- [41] Matthias Fitzi, Aggelos Kiayias, Giorgos Panagiotakos, and Alexander Russell. 2022. Ofelimos: Combinatorial Optimization Via Proof-of-Useful-Work: A Provably Secure Blockchain Protocol. In *Advances in Cryptology (CRYPTO)*.

- [42] Gapcoin. 2014. *Gapcoin*. Retrieved June 7, 2023 from <https://gapcoin.org/>
- [43] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2024. The Bitcoin Backbone Protocol: Analysis and Applications. *J. ACM* (apr 2024).
- [44] Michael R Garey and David S Johnson. 1979. *Computers and Intractability*. Vol. 174. freeman San Francisco.
- [45] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In *ACM SIGSAC Conference on Computer and Communications Security*.
- [46] B. Golden, L. Bodin, T. Doyle, and W. Stewart. 1980. Approximate Traveling Salesman Algorithms. *Operations Research* 28, 3 (1980), 694–711.
- [47] Carla Gomes and Richard Williams. 2006. Approximation Algorithms. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. 557–585.
- [48] Jens Groth. 2016. On the Size of Pairing-Based Non-Interactive Arguments. In *Advances in Cryptology (EUROCRYPT)*.
- [49] Richard Guy. 2004. *Unsolved problems in number theory*. Vol. 1. Springer Science & Business Media.
- [50] Mohamed Haouari, Mariem Mhiri, Mazen El-Masri, and Karim Al-Yafi. 2022. A novel proof of useful work for a blockchain storing transportation transactions. *Information Processing & Management* 59, 1 (2022).
- [51] David A. Harding and Peter Todd. 2015. *Opt-in Full Replace-by-Fee Signaling*. <https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki>
- [52] Godfrey Harold Hardy and Edward Maitland Wright. 1979. *An introduction to the theory of numbers*. Oxford university press.
- [53] Keld Helsgaun. 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research* 126, 1 (2000), 106–130.
- [54] K. Helsgaun. 2021. *LKH-3*. <http://akira.ruc.dk/~keld/research/LKH-3/>
- [55] Felix Hoffmann and Udo Kerschull. 2023. HEPchain: Novel Proof-of-Useful-Work blockchain consensus for High Energy Physics. arXiv:2304.13507 [cs.DC]
- [56] Stefan Hougardy and Xianghui Zhong. 2020. Hard to solve instances of the Euclidean Traveling Salesman Problem. *Mathematical Programming Computation* 13, 1 (Mar 2020), 51–74.
- [57] <https://bitcoin.stackexchange.com/>. 2012. *Nonce size*. <https://bitcoin.stackexchange.com/questions/1781/nonce-size-will-it-always-be-big-enough>
- [58] <https://bitcoin.stackexchange.com/>. 2018. *Shortest and Longest block interval time ever recorded in Bitcoin*. <https://bitcoin.stackexchange.com/questions/77783/shortest-and-longest-block-interval-time-ever-recorded-in-bitcoin>
- [59] Mahbub Hussain, Jordan J Bird, and Diego R Faria. 2019. A study on cnn transfer learning for image classification. In *Advances in Computational Intelligence Systems: UK Workshop on Computational Intelligence*.
- [60] Atalay M. Ileri, Halil I. Ozeran, Alper Gundogdu, Ahmet K. Senol, M. Yusuf Ozkaya, and Can Alkan. 2016. Coinami: A Cryptocurrency with DNA Sequence Alignment as Proof-of-work. arXiv:1602.03031 [cs.CE]
- [61] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. 2001. Which Problems Have Strongly Exponential Complexity? *J. Comput. System Sci.* 63, 4 (2001), 512–530.
- [62] intel.com. 2022. *APP Metrics for Intel Microprocessors*. <https://www.intel.com/content/dam/support/us/en/documents/processors/APP-for-Intel-Core-Processors.pdf>
- [63] Mohammadreza Ipchi Sheshgelani, Saeid Pashazadeh, and Pedram Salehpoor. 2023. Cooperative hybrid consensus with function optimization for blockchain. *Cluster Computing* 26, 6 (2023).
- [64] Markus Jakobsson and Ari Juels. 1999. Proofs of Work and Bread Pudding Protocols. In *Comm. and MM Security*.
- [65] Rie Johnson and Tong Zhang. 2013. Accelerating Stochastic Gradient Descent Using Predictive Variance Reduction. In *International Conference on Neural Information Processing Systems*.
- [66] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. 2018. Predicting the Computational Cost of Deep Learning Models. In *IEEE International Conference on Big Data*.
- [67] Julien Keuffer, Refik Molva, and Hervé Chabanne. 2018. Efficient Proof Composition for Verifiable Computation. In *Computer Security*.
- [68] Aggelos Kiayias and Giorgos Panagiotakos. 2015. Speed-Security Tradeoffs in Blockchain Protocols. *Cryptol. ePrint Arch.*, Paper 2015/1019.
- [69] Sunny King. 2013. *Primecoin: Cryptocur. with Prime Number Proof-of-Work*. <http://primecoin.io/bin/primecoin-paper.pdf>
- [70] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. 2018. xJsnark: A Framework for Efficient Verifiable Computation. In *S&P*.
- [71] Michał Król, Alberto Sonnino, Mustafa Al-Bassam, Argyrios G. Tasiopoulos, Etienne Rivière, and Ioannis Psaras. 2021. Proof-of-Prestige: A Useful Work Reward System for Unverifiable Tasks. *ACM Trans. Internet Technol.* 21, 2 (2021).
- [72] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Tr. Program. Lang. Sys.* 4, 3 (1982).

- [73] Boyang Li, Changhao Chenli, Xiaowei Xu, Taeho Jung, and Yiyu Shi. 2019. Exploiting Computation Power of Blockchain for Biomedical Image Segmentation. In *Conference on Computer Vision and Pattern Recognition Workshops*.
- [74] Boyang Li, Changhao Chenli, Xiaowei Xu, Yiyu Shi, and Taeho Jung. 2020. DLBC: A Deep Learning-Based Consensus in Blockchains for Deep Learning Services. arXiv:1904.07349 [cs.DC]
- [75] Wei Li. 2018. Adapting Blockchain Technology for Scientific Computing. arXiv:1804.08230 [cs.CR]
- [76] Henri Lifchitz. 1998. *Generalization of Euler-Lagrange theorem and new primality tests*. Retrieved April 11, 2023 from <http://www.primenumbers.net/Henri/us/NouvTh1us.htm>
- [77] Andrei Lihu, Jincheng Du, Igor Barjaktarevic, Patrick Gerzanic, and Mark Harvilla. 2020. A Proof of Useful Work for Artificial Intelligence on the Blockchain. arXiv:2001.09244 [cs.DC]
- [78] S. Lin and B. W. Kernighan. 1973. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research* 21, 2 (1973), 498–516.
- [79] Yuan Liu, Zhengpeng Ai, Shuai Sun, Shuangfeng Zhang, Zelei Liu, and Han Yu. 2020. Fedcoin: A peer-to-peer payment system for federated learning. In *Federated learning: privacy and incentive*. 125–138.
- [80] Yuan Liu, Yixiao Lan, Boyang Li, Chunyan Miao, and Zhihong Tian. 2021. Proof of Learning (PoLe): Empowering Neural Network Training with Consensus Building on Blockchains. *Comput. Netw.* 201, C (2021).
- [81] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. 2014. On the Computational Efficiency of Training Neural Networks. In *International Conference on Neural Information Processing Systems*.
- [82] Angelique Faye Loe and Elizabeth A. Quaglia. 2018. Conquering Generals: an NP-Hard Proof of Useful Work. In *Work. on Cryptocurrencies and Blockchains for Distrib. Syst.*
- [83] Nancy A. Lynch. 1996. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc.
- [84] Uroš Maleš, Dušan Ramljak, Tatjana Jakšić Krüger, Tatjana Davidović, Dragutin Ostojić, and Abhay Haridas. 2023. Controlling the Difficulty of Combinatorial Optimization Problems for Fair Proof-of-Useful-Work-Based Blockchain Consensus Protocol. *Symmetry* 15, 1 (2023).
- [85] Andrea Merlina. 2019. BlockML: A Useful Proof of Work System Based on Machine Learning Tasks. In *ACM International Middleware Conference Doctoral Symposium*.
- [86] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. 2014. Permacoin: Repurposing Bitcoin Work for Data Preservation. In *IEEE Symposium on Security and Privacy*.
- [87] Anshul Mittal and Swati Aggarwal. 2020. Hyperparameter optimization using sustainable proof of work in blockchain. *Frontiers in Blockchain* 3 (2020), 23.
- [88] Satoshi Nakamoto. 2009. *Bitcoin: A Peer-to-Peer Electronic Cash System*. www.bitcoin.org
- [89] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. 2016. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press.
- [90] Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. 2019. Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities. *IEEE Access* 7 (2019).
- [91] University of Heidelberg. 1995. *TSPLIB Format*. <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95>
- [92] University of Waterloo. 2021. *World Traveling Salesman Problem*. <https://www.math.uwaterloo.ca/tsp/world/>
- [93] Carlos G. Oliver, Alessandro Ricottone, and Pericles Philippopoulos. 2017. Proposal for a fully decentralized blockchain and proof-of-work algorithm for solving NP-complete problems. arXiv:1708.09419 [cs.DC]
- [94] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *IEEE Symposium on Security and Privacy*.
- [95] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology (EUROCRYPT)*.
- [96] Rafael Pass and Elaine Shi. 2017. FruitChains: A Fair Blockchain. In *ACM Symposium on Principles of Distributed Computing*.
- [97] Pericles Philippopoulos, Alessandro Ricottone, and Carlos G. Oliver. 2020. Difficulty Scaling in Proof of Work for Decentralized Problem Solving. *Ledger* 5 (2020).
- [98] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: scalable off-chain instant payments.
- [99] Chao Qiu, Xiaofei Wang, Haipeng Yao, Jianbo Du, F. Richard Yu, and Song Guo. 2021. Networking Integrated Cloud–Edge–End in IoT: A Blockchain-Assisted Collective Q-Learning Approach. *IEEE Internet of Things Journal* 8, 16 (2021).
- [100] Xidi Qu, Shengling Wang, Qin Hu, and Xiuzhen Cheng. 2021. Proof of Federated Learning: A Novel Energy-Recycling Consensus Algorithm. *IEEE Transactions on Parallel and Distributed Systems* 32, 8 (2021), 2074–2085.
- [101] Gerhard Reinelt. 1991. TSPLIB—A Traveling Salesman Problem Library. *ORSA Journal on Computing* 3, 4 (1991).
- [102] Meni Rosenfeld. 2011. Analysis of Bitcoin Pooled Mining Reward Systems. arXiv:1112.4980 [cs.DC]
- [103] Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis, II. 1977. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM J. Comput.* 6, 3 (1977), 563–581.

- [104] Nicolas Roux, Mark Schmidt, and Francis Bach. 2012. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. In *Advances in Neural Information Processing Systems*.
- [105] Stuart Russell and Peter Norvig. 2021. *Artificial Intelligence: A Modern Approach* (4rd ed.). Prentice Hall Press.
- [106] Mahmoud Salhab and Khaleel Mershad. 2023. Proof of Deep Learning: Approaches, Challenges, and Future Directions. arXiv:2308.16730 [cs.CR]
- [107] Andrzej Schinzel and Waclaw Sierpiński. 1958. Sur certaines hypotheses concernant les nombres premiers. *Acta Arithmetica* 4, 3 (1958), 185–208.
- [108] Naoki Shibata. 2019. Proof-of-Search: Combining Blockchain Consensus Formation With Solving Optimization Problems. *IEEE Access* 7 (2019).
- [109] A. Shoker. 2017. Sustainable blockchain through proof of exercise. In *IEEE Network Computing and Applications*.
- [110] Yonatan Sompolsinsky and Aviv Zohar. 2015. Secure High-Rate Transaction Processing in Bitcoin. In *Financial Cryptography*.
- [111] Gilbert Strang. 2023. *Introduction to Linear Algebra* (6 ed.). CUP.
- [112] Willa Ariela Syafruddin, Sajjad Dadkhah, and Mario Köppen. 2019. Blockchain Scheme Based on Evolutionary Proof of Work. In *IEEE Congress on Evolutionary Computation*.
- [113] Yingjie Tian and Yuqi Zhang. 2022. A comprehensive survey on regularization strategies in machine learning. *Information Fusion* 80 (2022).
- [114] Milan Todorović, Luka Matijević, Dušan Ramljak, Tatjana Davidović, Dragan Urošević, Tatjana Jakšić Jakšić Krüger, and Đorđe Jovanović. 2022. Proof-of-Useful-Work: Blockchain Mining by Solving Real-Life Optimization Problems. *Symmetry* 14, 9 (2022).
- [115] Marko Vukolić. 2016. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *Open Problems in Network Security*.
- [116] Michael Walfish and Andrew J. Blumberg. 2015. Verifying Computations without Reexecuting Them. *Comm. ACM* 58, 2 (2015).
- [117] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. 2023. SoK: DAG-Based Blockchain Systems. *ACM Comp. Sur.* 55, 12 (2023).
- [118] Yunkai Wei, Zixian An, Supeng Leng, and Kun Yang. 2023. Evolved PoW: Integrating the Matrix Computation in Machine Learning Into Blockchain Mining. *IEEE Internet of Things Journal* 10, 8 (2023).
- [119] Bitcoin Wiki. 2022. *Bitcoin's Difficulty*. <https://en.bitcoin.it/wiki/Difficulty>
- [120] Wikipedia. 2023. *Proof of work*. https://en.wikipedia.org/wiki/Proof_of_work
- [121] Andrew Wiles. 1995. Modular Elliptic Curves and Fermat's Last Theorem. *Annals of Mathematics* 141, 3 (1995), 443–551.
- [122] Ryan Williams. 2005. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* 348, 2 (2005), 357–365. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004).
- [123] Virginia Vassilevska Williams. 2015. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis. In *Int. Symp. on Parameterized and Exact Comp.*
- [124] Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. 2020. A Survey of Distributed Consensus Protocols for Blockchain Networks. *IEEE Communications Surveys and Tutorials* 22, 2 (2020).
- [125] Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. In *Int. Conf. on Learning Representations*.
- [126] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. 2017. REM: Resource-Efficient Mining for Blockchains. In *USENIX Conference on Security*.
- [127] Weilin Zheng, Xu Chen, Zibin Zheng, Xiapu Luo, and Jiahui Cui. 2020. AxeChain: A Secure and Decentralized blockchain for solving Easily-Verifiable problems. arXiv:2003.13999 [cs.DC]

A MATRIX SIZE FOR GIVEN COMPUTING POWER AND TIME

The goal is (a) to estimate the instance size required to propose a k-OV solution every 10 minutes on average, like in Bitcoin, and (b) to demonstrate that the required size is impractical. While there is no clear unique way to demonstrate (b), every time we select an underlying assumption, we choose the assumption resulting in a smaller instance size i.e. which is unfavorable to (b). This allows us to calculate a lower bound on the actual instance size, and show its impracticality. Under realistic assumptions, that size will likely be even more significant.

To demonstrate (b), we start by considering a mining pool and estimate the following: (1) the computing power of the mining pool measured in FLOPS instead of hashes/s, and (2) the expected

time for the pool to solve a k-OV task. To address step (1), we retrieved updated performance data about CPUs and ASICs. The latest generation of Intel processors executes roughly $800 \cdot 10^9$ FLOPS [62], while the latest AntMiner, a popular mining ASIC, has a hashrate of $190 \cdot 10^{12}$ hashes/s [14]. Directly converting hashes/s into FLOPS leads to an improvement of three orders of magnitude per computing unit. Instead, we select the conservative case of an equal number of computing units, an assumption unfavorable to point (b).

At the time of writing, Bitcoin’s biggest mining pool has 100 Ehashes/s³. Considering an AntMiner of the latest generation, we estimate that such a mining pool comprises roughly $5 \cdot 10^5$ ASICs. This corresponds to a mining pool of 400 PFLOPS.

As step (2), we now estimate the expected proposal time. In Bitcoin, the whole network proposes a new block every 10 minutes on average. Individual mining pools, on the other hand, propose blocks much less frequently. In this example, we assume that the proposal frequency of the mining pool used in our calculations is the same as the proposal frequency of the whole network. This simplification is unfavorable to the hypothesis in (b).

Finally, by using the upper bound $(ct)^{1/3}$ on the size of matrices, we estimate that a mining pool of 400 PFLOPS requires matrices of dimension $n \approx 6.2 \cdot 10^6$ to solve a k-OV instance in 600 seconds.

³<https://btc.com/stats/pool>