

CASH: Shapley Value Computation with Cache Optimization

Jiajun Tang
Zhejiang University
jiajuntang@zju.edu.cn

Xiaokai Mao
Zhejiang University
xiaokaimao@zju.edu.cn

Ning Liu
Zhejiang University
ninliu@umich.edu

Jinfei Liu*
Zhejiang University
jinfeiliu@zju.edu.cn

Kui Ren
Zhejiang University
kuiren@zju.edu.cn

ABSTRACT

In recent years, the Shapley value has become the de facto standard for equitable attribution in data analytics, such as data valuation and model interpretability. Since exact computation entails an exponential complexity of $O(2^n)$, sampling-based approximation algorithms are widely adopted. However, these methods treat utility functions as stateless black boxes, leading to a critical system-level inefficiency: the redundant and costly evaluation of identical coalitions that recur during sampling. To address this bottleneck, we propose CASH, an algorithm-agnostic Caching framework that accelerates existing Shapley value approximation algorithms by strategically storing and reusing intermediate coalition utility computations. CASH leverages a high-performance Direct Mapping architecture tailored for Shapley value approximation to cache coalition utility results, enabling significant speedups without introducing any additional approximation error. We integrate CASH with major approximation algorithms and evaluate the performance across diverse data analytics tasks. Experimental results demonstrate that CASH consistently accelerates widely used approximation algorithms, reducing total computation time by 8% to 29% depending on the underlying sampling strategy. This efficiency gain is achieved without introducing additional approximation error beyond the underlying estimator, improving the efficiency of Shapley value-based data analytics pipelines.

PVLDB Reference Format:

Jiajun Tang, Xiaokai Mao, Ning Liu, Jinfei Liu, and Kui Ren. CASH: Shapley Value Computation with Cache Optimization. PVLDB, 19(10): 2777 - 2789, 2026.

doi:10.14778/3828612.3828631

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/ZJU-DIVER/OpenSV>.

1 INTRODUCTION

The Shapley value has established itself as the cornerstone of equitable attribution in modern data analytics pipelines [23]. Its widespread adoption stems from a rigorous axiomatic foundation: it

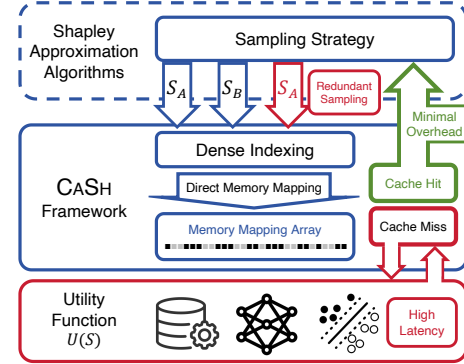


Figure 1: Overview of CASH. Acting as a transparent intermediary layer between sampling algorithms and the utility function evaluation, it intercepts coalition queries using a high-performance Dense Indexing mechanism. Redundant queries (e.g., S_A) are served instantly via Direct Mapping, bypassing the expensive utility function evaluation.

is the unique allocation scheme satisfying *efficiency*, *symmetry*, *dummy player*, and *linearity* [34]. This theoretical robustness has transformed the Shapley value from a game-theoretic concept into a core database primitive, essential for a wide spectrum of data management tasks [40].

Consequently, it is increasingly integrated into modern systems to support equitable data valuation in marketplaces and collaborative analytics [4, 7–9, 12, 16, 24, 32, 43], utility-based data selection for efficient training and debugging [12, 15, 21, 23, 27, 33, 37, 39], and provenance-aware query explanation [3, 14, 25, 29]. However, the widespread deployment of these applications faces a severe bottleneck: the prohibitive cost of computing Shapley values.

The exact computation of Shapley values is known to be #P-hard, as it requires evaluating exponentially many coalition utilities. Existing sampling-based approximation algorithms, such as Monte Carlo permutation sampling [6, 12], Antithetic sampling [30], Stratified sampling [11], SVARM [20], and GELS-Shapley [22], make Shapley-based analytics tractable by reducing the number of sampled coalitions. This paper is complementary to these estimators: rather than changing the sampling distribution or improving statistical convergence, we focus on an overlooked system-level inefficiency, namely that identical coalitions generated during sampling are often evaluated repeatedly.

*Jinfei Liu is the corresponding author.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 19, No. 10 ISSN 2150-8097.

doi:10.14778/3828612.3828631

Motivations. Shapley value approximation algorithms generally comprise two categories: *permutation-based sampling* [31] and *subset-based sampling* [26], as summarized in Table 1. Permutation-based sampling is the dominant approach in widely adopted methods [40]. Its primary advantage is high update efficiency. By sequentially adding players in a random order, it can efficiently estimate marginal contributions. However, this approach suffers from a critical structural inefficiency. It operates in a vast permutation space of size $n!$, which is significantly larger than the coalition space of size 2^n . This mismatch leads to frequent collisions. Different permutations often generate the identical coalition S . Consequently, the algorithm incurs significant computational waste by repeatedly evaluating the utility $U(S)$ for the same coalition. Our analysis in Section 3.2 shows that nearly 15% of utility calls are redundant in Monte Carlo experiments.

In contrast, subset-based sampling operates directly within the coalition space of size 2^n . This strategy naturally avoids the redundancy issues of permutation-based methods. However, it suffers from significant practical limitations. First, the estimation process introduces implementation complexity. Unlike straightforward averaging in permutation methods, subset-based sampling requires complex weighting schemes to ensure the result converges to the true Shapley value. Second, it exhibits low update efficiency. To compute a marginal contribution, subset-based sampling typically requires two independent utility evaluations. It lacks the ability to use fast incremental updates that are characteristic of permutation sampling.

This trade-off raises a fundamental research question in the Shapley value approximation. *Can we design a high-performance mechanism to synergize high update efficiency of permutation-based sampling while strictly avoiding redundancy as in subset-based sampling?*

Contributions. In this paper, we propose CASH, an algorithm-agnostic, plug-and-play **Caching** framework that accelerates existing **Shapley** value approximation algorithms, as shown in Figure 1. The figure depicts CASH as a transparent intermediary layer between the sampling procedure and the utility function evaluation. It intercepts the stream of coalition queries (e.g., S_A, S_B, S_A). When a redundant sampling occurs, such as the recurring S_A shown in the figure, CASH leverages Direct Mapping and Dense Indexing mechanisms to retrieve the stored value instantly. Crucially, CASH requires no modifications to the underlying approximation algorithm or the utility function implementation, making it immediately deployable on existing algorithms. By bridging the gap between the two sampling paradigms, CASH retains the efficient incremental updates of permutation methods while effectively converting implicit permutation redundancies into explicit cache hits.

The core innovation of CASH lies in its high-performance **Direct Mapping** architecture. We first identify that standard hash tables incur non-negligible micro-architectural overheads (e.g., hash computation and collision resolution) that limit acceleration. To achieve minimal lookup latency, we propose using direct memory addressing. A naive implementation would naturally treat a coalition S as a binary vector and use its integer value as the memory offset. While this offers $O(1)$ access, it inevitably suffers from the memory explosion problem. Since the binary indices are sparse, mapping the

Table 1: Comparison of Sampling Strategies.

Feature / Metric	Permutation Based	Subset Based	CASH (Ours)
Sampling Space	$n!$	2^n	$n! \rightarrow 2^n$
Update Efficiency	✓ High	✗ Low	✓ High
Redundancy	✗ High	✓ Low	✓ Low
Implementation	✓ Low	✗ High	✓ Low*

* CASH functions as transparent middleware, requiring minimal code changes.

entire space requires allocating a contiguous memory block of size 2^n , which is infeasible for moderate n (e.g., $n = 50$). To address this problem, we leverage the structural insight that redundant computations are heavily concentrated in either small or large coalitions. Consequently, CASH restricts caching exclusively to these regions. By integrating this strategy with a **Dense Indexing** scheme, we compress the specific target coalitions into a contiguous memory block. This approach reduces the memory footprint to a manageable scale of $O(\sum_{l=0}^L \binom{n}{l})$.

We implement CASH and integrate it with major Shapley value approximation algorithms. Our experiments on diverse data analytics tasks show that CaSh reduces total computation time by 8% to 29% across different approximation algorithms and datasets, with larger gains for sampling strategies that frequently revisit small and large coalitions like GELS [22], while preserving the approximation accuracy of the underlying algorithms. Furthermore, we show that CASH is not limited to data valuation; it naturally extends to SHAP-based model interpretability [26], where feature subsets play the role of coalitions. Given that feature spaces are typically small, the duplicate rate is even higher, making caching exceptionally effective. Our main contributions are summarized as follows.

- We identify and quantify redundant coalition evaluations as a widespread yet overlooked structural inefficiency in existing sampling-based Shapley value approximation algorithms.
- We propose CASH, which leverages a high-performance Direct Mapping architecture to eliminate redundant utility function evaluations. Guided by the distribution of recurring coalitions in Shapley value approximation sampling, we employ Coalition Size Narrowing to focus on high-redundancy regions, complemented by a Dense Indexing scheme for compact memory management.
- We demonstrate through extensive experiments that CASH delivers universal speedups across diverse algorithms and datasets without introducing additional approximation error beyond the underlying estimator, and shows competitive performance in SHAP-style interpretability tasks.

The remainder of this paper is organized as follows. Section 2 reviews the existing literature on data analytics. Section 3 establishes the theoretical foundations, formalizes the re-evaluation problem, and quantifies the duplicate rate in Monte Carlo sampling. Section 4 details the architectural design of CASH, introducing our core technical innovations. Section 5 presents a comprehensive experimental evaluation, demonstrating the universality of our framework across diverse algorithms and datasets. Finally, Section 6 summarizes our contributions and concludes the paper.

2 RELATED WORK

Our work focuses on accelerating the Shapley value approximation for data analytics, including data valuation and model interpretability [23]. We review related efforts in these domains, specifically highlighting the system-level inefficiencies inherent in current approximation algorithms.

Shapley Value in Data Analytics. The Shapley value [34] has become a foundational primitive for attribution in modern data analytics [10, 12, 26, 32, 36, 42]. Data valuation and model interpretability differ in application goals but share the same mathematical kernel, enabling common optimization techniques.

In data valuation, it quantifies the contribution of records or providers for equitable compensation in data pricing [24, 40], noisy or adversarial data detection in data quality assurance [16], and coresets selection by pruning low-value contributors [12, 37]. In model interpretability, SHAP [26] treats features as players; since the combinatorial problem is unchanged, CASH can also accelerate interpretability tasks, especially with moderate feature counts.

Shapley Value Approximation. To circumvent the #P-hard complexity of exact computation, a rich line of work has developed sampling-based approximation algorithms. These methods typically estimate the Shapley value as an expectation of marginal contributions, differing primarily in their sampling strategies.

The foundational Monte Carlo Permutation Sampling [6, 12, 28] estimates contributions by averaging over random player permutations. To accelerate convergence, variance reduction techniques have been introduced. Antithetic Sampling [30] pairs every sampled permutation with its reverse to induce negative correlation, while Complementary Contribution schemes [41] explicitly evaluate coalitions alongside their complements.

Other approaches focus on the structure of the coalition space. Stratified Sampling [12] allocates fixed sampling budgets to different coalition sizes to handle non-uniform variance. Building on this, recent methods like SVARM Stratified [20] and GELS-Shapley [22] employ sophisticated weighting schemes or regression-based estimators to prioritize “informative” coalitions.

However, the primary focus of these methods is on reducing the number of samplings required for statistical convergence. They typically treat the utility function as a stateless “black box.” Consequently, they do not address the system-level inefficiency of *redundant computations*, where the same coalition can be resampled and re-evaluated, either through random collision or exhaustive enumeration of small strata. Our work complements these algorithmic advancements by adding a stateful caching layer to reduce redundant utility evaluations.

3 PRELIMINARIES AND MOTIVATION

In this section, we establish the formal foundations for our work. We begin in Section 3.1 by defining the Shapley value and highlighting the prohibitive computational cost of exact solutions. In Section 3.2, we then identify a critical system-level inefficiency shared by approximation algorithms: the stateless re-evaluation of recurring coalitions. By rigorously quantifying this redundancy and demonstrating its severity across typical problem scales in data valuation and model interpretability, we provide the empirical and theoretical motivation for the CASH framework.

3.1 The Shapley Value Framework

In the context of data analytics, the Shapley value provides a principled framework for quantifying the contribution of each data contributor. This is formally modeled as a cooperative game. A cooperative game is defined by a finite set of players $N = \{p_1, p_2, \dots, p_n\}$, which in our setting correspond to data sources, records, or features; and a utility function $U : 2^N \rightarrow \mathbb{R}$. The utility function $U(S)$ evaluates the collective utility of any subset $S \subseteq N$ (known as a *coalition*), with $U(\emptyset) = 0$. Throughout this paper, we treat the utility function as deterministic. For inherently stochastic evaluators (e.g., training neural networks via SGD), we ensure determinism by fixing random seeds. This practice improves reproducibility and reduces the influence of algorithmic randomness on the measured marginal contributions. For database workloads, $U(S)$ typically represents the utility of query answers derived from coalition S or the performance of a machine learning model trained exclusively on coalition S .

The Shapley value $\mathcal{SV}_i$ is the unique allocation that fairly distributes $U(N)$ by accounting for the marginal contribution of p_i to every possible coalition. Formally,

$$\mathcal{SV}_i = \frac{1}{n} \sum_{S \subseteq N \setminus \{p_i\}} \binom{n-1}{|S|} (U(S \cup \{p_i\}) - U(S)). \quad (1)$$

This equation naturally leads to subset-based sampling, where subsets S are drawn according to a specific distribution.

An equivalent and often computationally useful formulation expresses the Shapley value as an average over all permutations of the players. Let $\Pi(N)$ be the set of all $n!$ permutations of N . For a given permutation $\sigma \in \Pi(N)$, let $[\sigma]_{p_i-1}$ denote the set of players that precede p_i in σ . The Shapley value can then be written as

$$\mathcal{SV}_i = \frac{1}{n!} \sum_{\sigma \in \Pi(N)} (U([\sigma]_{p_i-1} \cup \{p_i\}) - U([\sigma]_{p_i-1})). \quad (2)$$

This equation serves as the foundation for permutation-based sampling. By iterating through random permutations, this method efficiently updates the Shapley values of all players simultaneously.

The Shapley value is theoretically distinguished as the *unique* allocation scheme satisfying the canonical set of fairness axioms in cooperative game theory. This axiomatic guarantee makes it the preferred choice for fairness-critical tasks such as data pricing and influence analysis.

3.2 The Computational Challenge: Redundancy

Despite its theoretical appeal, practical deployment of the Shapley value is hindered by the #P-Hard complexity, which requires evaluating $U(S)$ for all 2^n coalitions. To make this tractable, sampling-based approximation algorithms (e.g., Monte Carlo, Stratified Sampling, GELS-Shapley) are widely adopted.

While these algorithms differ in sampling strategies, they share a critical system-level inefficiency: they frequently generate identical coalitions across different iterations and needlessly recompute $U(S)$. In data analytics scenarios, where $U(S)$ involves computationally intensive operations, ranging from complex query execution to model training, this redundancy wastes significant computational resources.

Algorithm 1: Monte Carlo Permutation Sampling

```
1  $\mathcal{SV}_i \leftarrow 0$ ;  
2 for  $t \leftarrow 1$  to  $n_p$  do  
3    $\pi \leftarrow \text{RandomPermutation}(N)$ ;  
4    $S \leftarrow \emptyset$ ;  $v_{\text{old}} \leftarrow 0$ ;  
5   foreach  $p \in \pi$  do  
6      $S \leftarrow S \cup \{p\}$ ;  
7      $v_{\text{new}} \leftarrow U(S)$ ; // May repeat previous evaluations  
8      $\mathcal{SV}_p \leftarrow v_{\text{new}} - v_{\text{old}}$ ;  
9      $v_{\text{old}} \leftarrow v_{\text{new}}$ ;  
10  $\mathcal{SV}_i \leftarrow \mathcal{SV}_i / n_p$ ;  
11 return  $\mathcal{SV}_i$ .
```

An intuitive shortcut to mitigate this redundancy might be to directly generate specific permutation prefixes. However, to ensure an unbiased estimation of the Shapley value, any generated prefix S for a player i must follow the probability distribution induced by random permutations: $\Pr(S) = \frac{|S|!(n-|S|-1)!}{n!}$. Selectively truncating intermediate sizes or altering this distribution may introduce systematic approximation bias.

To quantify this redundancy, we analyze the classic Monte Carlo method (Algorithm 1) as a representative example. Given a budget of T_U total utility function evaluations, the algorithm generates $n_p = T_U/n$ permutations. Since each permutation creates exactly one prefix of length l for every $l \in \{1, \dots, n\}$, the algorithm effectively draws n_p samples from the pool of size- l coalitions. Let $\{S_{l,1}, \dots, S_{l,n_p}\}$ denote this sequence. The number of *distinct* coalitions sampled of size l is a random variable, denoted as X_l .

Formally, let $n_l = \binom{n}{l}$ be the total number of possible coalitions of size l , and let $S'_{l,1}, \dots, S'_{l,n_l}$ be an enumeration of them. We define an indicator variable $X_{l,i} = 1$ if coalition $S'_{l,i}$ appears in the sample, and 0 otherwise. Since the probability of a specific coalition appearing in a random permutation prefix is $1/n_l$, we have $E[X_{l,i}] = 1 - (1 - \frac{1}{n_l})^{n_p}$. The expected number of distinct coalitions of size l is then

$$E[X_l] = \sum_{i=1}^{n_l} E[X_{l,i}] = n_l \left(1 - \left(1 - \frac{1}{n_l} \right)^{n_p} \right). \quad (3)$$

Summing over all coalition sizes, the total expected number of distinct utility evaluations is $n_{\text{dist}}(n_p) = \sum_{l=1}^n E[X_l]$. The expected number of duplicate evaluations is therefore

$$n_{\text{dup}}(n_p) = n \cdot n_p - n_{\text{dist}}(n_p). \quad (4)$$

To assess the magnitude of this redundancy in representative settings, we parameterize this model with problem scales that arise in common Shapley-based analytics workloads. In the context of model interpretability, where players are features, our analysis of 210 datasets from the UCI repository [17] reveals that 72.9% involve 30 or fewer features. Similarly, in data valuation, players typically represent distinct data partitions or organizations (e.g., in federated learning) rather than individual records, also resulting in moderate player counts.

Based on this observation, we apply our model to a representative problem size of $n = 30$ with a sampling budget of $n_p = 1,000$ permutations (totaling 30,000 queries). Plugging these values into our model reveals that $n_{\text{dup}} \approx 4,429$, meaning that 14.8% of all utility

Algorithm 2: CASH

Input : Target Coalition S , Original Utility Function $U(\cdot)$,
Cache store C (structure depends on strategy)

Output: Utility value v

```
1  $\text{CachedEvaluator}(S)$   
2 begin  
3    $v \leftarrow \text{Get}(C, S)$ ;  
4   if  $v$  is null then  
     // Cache Miss: Expensive evaluation  
     // Invoke original utility function  
      $v \leftarrow U(S)$ ;  
     // Store result for future reuse  
      $\text{Insert}(C, S, v)$ ;  
5   return  $v$ .
```

evaluations are redundant. This implies that nearly 4,500 expensive model training or inference steps are wasted on recomputing known results.

4 CASH

In this section, we present the design and implementation of CASH. We begin in Section 4.1 by establishing the decoupling architecture that allows the framework to function as a transparent middleware. We then address the performance limitations of traditional caching in Section 4.2 by introducing our core innovation, **Direct Mapping**. To resolve the associated memory constraints, we employ a two-pronged strategy: **Coalition Size Narrowing** (Section 4.3) to select high-value targets and **Dense Indexing** (Section 4.4) to compactly map them. Finally, in Section 4.5, we analyze the theoretical bounds of **Hybrid Caching** to validate the optimality of our design choice.

4.1 Algorithm-Agnostic Caching Layer

We first introduce the fundamental concept of CASH as a transparent intermediary layer that intercepts interactions between the sampling algorithm and the evaluation of the utility function. Our architectural premise is that redundancy management is orthogonal to statistical approximation. By decoupling these concerns, CASH functions as a general caching layer. It can be seamlessly integrated into the existing Shapley value computation pipeline without requiring modifications to either the upstream sampling logic or the downstream utility implementation, allowing it to be integrated with minimal changes to existing implementations.

Specifically, we implement this layer as a lightweight intermediary function, which we refer to as the **cached utility evaluator**. This evaluator acts as a proxy between the sampling algorithm and the actual, often expensive, evaluation of utility function $U(\cdot)$. Whenever the sampling algorithm needs the utility of a coalition S , it calls this evaluator rather than evaluating $U(S)$ directly.

As outlined in Algorithm 2, the cached utility evaluator operates with a minimal execution footprint. Upon receiving a query for coalition S , it first attempts a retrieval from the global cache store C . If a valid value is found, it is returned immediately with negligible latency, effectively bypassing the expensive computation (line 3).

Only when the retrieval yields null (a cache miss) does the evaluator invoke the original utility function $U(S)$ and Insert the result for future reuse (line 4). This guarantees that $U(S)$ is executed at most once for any cached coalition, eliminating redundancy while maintaining the original algorithm’s convergence properties.

The most straightforward implementation for this internal data structure is a standard hash table. While theoretically offering $O(1)$ average lookup time, it suffers from significant micro-architectural overheads, specifically hash computation, collision resolution, and dynamic memory allocation. These costs become dominant when the utility function $U(S)$ is not extremely expensive. Furthermore, as we verified in Section 5, the management overhead of a hash table often outweighs the computational savings for the vast majority of intermediate-sized coalitions where duplicate probabilities are low. To overcome these limitations, we propose **Direct Mapping**. This strategy replaces hash-based lookup with arithmetic addressing. We detail this core mechanism below.

4.2 Direct Mapping

To eliminate the overhead of hash tables, our approach pre-allocates a contiguous memory block to store utility values. It establishes a unique link between a coalition S and a fixed memory index. This design completely bypasses the expensive hash calculations and collision checks found in standard storage methods.

The Naive Mechanism. The most intuitive implementation interprets a coalition $S \subseteq N$ directly as a binary bitmask. We define the binary encoding index $M_{\text{naive}}(S)$ as

$$M_{\text{naive}}(S) = \sum_{p_i \in S} 2^{i-1}. \quad (5)$$

Using this index, the system locates the utility value $U(S)$ via arithmetic address calculation, calculating the memory offset $\mathcal{A}(S)$ from a base address $\mathcal{A}_{\text{base}}$

$$\mathcal{A}(S) = \mathcal{A}_{\text{base}} + M_{\text{naive}}(S) \times \text{sizeof}(\text{entry}). \quad (6)$$

This design fundamentally differentiates Direct Mapping from a hash table. Although a standard hash table also relies on such a canonical integer representation (essentially M_{naive}) to serve as the lookup key, it must subsequently subject this key to a hash function, resolve collisions, and chase pointers. Direct Mapping bypasses these intermediate layers, using the key directly as the memory offset. This ensures consistent, minimal access latency regardless of cache occupancy.

Furthermore, this static addressing scheme supports lock-free concurrency. Since the utility function $U(S)$ is deterministic, simultaneous updates to the same cache entry are inherently safe. Even if multiple threads race to write the entry, they simply overwrite it with the identical value, eliminating the need for complex synchronization locks. This constitutes a “benign race condition” that requires no synchronization. We empirically validate this scalability advantage in Section 5.7. Conversely, a concurrent hash table typically demands expensive locking mechanisms (e.g., mutexes) to maintain structural integrity during resizing or collision handling, introducing significant contention overhead in multi-core environments.

The Challenge: Memory Explosion. However, this naive implementation faces a critical barrier: the Memory Explosion problem.

Since the binary index $M_{\text{naive}}(S)$ ranges from 0 to $2^n - 1$, this approach implicitly requires a contiguous memory block of size 2^n . The scale of this requirement is prohibitive. For a moderate problem size of $n = 30$, caching utility values (e.g., as 8-byte floats) already consumes 8 GiB of memory. While this might fit in RAM, the exponential growth renders the approach quickly intractable. Merely increasing n to 50 requires approximately 8 PiB (petabytes) of memory, an amount that far exceeds the capacity of any single modern server. Consequently, despite its latency benefits, naive Direct Mapping is impractical for moderate or large problem sizes.

To make Direct Mapping practical, we need to limit its memory footprint. We address this challenge by combining two complementary strategies:

- (1) **Coalition Size Narrowing (Section 4.3).** We theoretically demonstrate that caching is only beneficial for a small subset of “high-probability” coalitions, reducing the logical scope from 2^n to a manageable scale.
- (2) **Dense Indexing (Section 4.4).** We propose a compact indexing scheme $M_{\text{dense}}(S)$ to tightly pack these sparse coalitions into a pre-allocated block, eliminating memory fragmentation.

4.3 Coalition Size Narrowing

Given the impossibility of caching the entire 2^n coalition space, we must determine which coalitions are most beneficial to cache.

Skewed Sampling Density. Monte Carlo sampling generates coalitions of size l with uniform probability. However, the number of unique coalitions of size l is determined by $\binom{n}{l}$. This creates a disparity in sampling density.

- **High-Density Regimes.** When l is close to 0 or n , the coalition space is small. Even a limited sampling budget covers a significant fraction of the total subspace, which causes frequent repeated evaluations of the same coalition.
- **Sparse Regimes.** When l is close to $n/2$, the combinatorial space is large. For instance, $\binom{30}{15}$ exceeds 10^8 . Given a typical budget of 10^4 samples, the sampling density is low. Therefore, the likelihood of sampling the same coalition twice is minimal.

This structural imbalance dictates our strategy: caching is most cost-effective near the extremes of the coalition spectrum. Storing these high-probability coalitions tends to yield a higher cache hit rate, whereas allocating memory for the vast, sparsely sampled middle region incurs additional overhead with limited returns.

PROPOSITION 1. *For a fixed number of permutations $n_p \geq 1$, the expected number of duplicate evaluations $n_{\text{dup}}(n_p, l)$ is theoretically non-increasing with respect to the subspace size $n_l = \binom{n}{l}$. Treating n_l as a continuous variable for analysis, we have $\frac{\partial n_{\text{dup}}}{\partial n_l} \leq 0$.*

PROOF. Computing the partial derivative of $n_{\text{dup}}(n_p, l)$ with respect to n_l yields

$$\frac{\partial n_{\text{dup}}}{\partial n_l} = \underbrace{\frac{n_p + n_l - 1}{n_l - 1} \left(1 - \frac{1}{n_l}\right)^{n_p}}_{f(n_p, n_l)} - 1.$$

To prove $\frac{\partial n_{\text{dup}}}{\partial n_l} \leq 0$, it suffices to show $f(n_p, n_l) \leq 1$. We analyze the monotonicity of f with respect to n_p . Taking the partial derivative of $\ln f$

$$\frac{\partial \ln f}{\partial n_p} = \frac{1}{n_p + n_l - 1} + \ln \left(1 - \frac{1}{n_l} \right).$$

Using the inequality $\ln(1 - x) < -x$ for $x \in (0, 1)$, let $x = 1/n_l$, we have $\ln(1 - \frac{1}{n_l}) < -\frac{1}{n_l}$. Since $\frac{1}{n_p + n_l - 1} \leq \frac{1}{n_l}$ for $n_p \geq 1$, the derivative $\frac{\partial \ln f}{\partial n_p}$ is strictly negative.

Thus, $f(n_p, n_l)$ is a decreasing function of n_p . Noting that $f(1, n_l) = 1$, we conclude that $f(n_p, n_l) \leq 1$ for all $n_p \geq 1$. Consequently, $\frac{\partial n_{\text{dup}}}{\partial n_l} \leq 0$. $\square$

This theoretical result validates our intuition: as the combinatorial space $\binom{n}{l}$ explodes, the probability of sampling collisions becomes small. For instance, with $n = 30$ and $n_p = 1,000$, our model predicts $n_{\text{dup}} \approx 970$ for size $l = 1$, but this drops precipitously to $n_{\text{dup}} \approx 0.017$ for $l = 10$. In the latter regime, caching is unlikely to pay off because the indexing and storage overhead is not amortized by cache hits.

Optimizing the Cache Limit. This analysis directly motivates our Coalition Size Narrowing strategy. We introduce a **cache limit** L and restrict our Direct Mapping cache to only the most “cache-worthy” coalitions: those with size $|S| \leq L$ or $|S| \geq n - L$. This focused approach allows us to concentrate our memory resources where they are expected to provide the largest runtime benefit, while completely avoiding the overhead of managing a cache for sizes where hits are vanishingly rare.

However, determining the precise value of L involves a trade-off: increasing L captures more duplicates but incurs higher linear overheads. To identify the optimal configuration, we formulate a cost model to cast the selection of L as a utility maximization problem.

For given grand player coalition N and its size $n = |N|$, let t_U denote the time cost of a utility function evaluation, t_D be the amortized time cost of a Direct Mapping cache access (reading or storing a pair $(S, U(S))$), and t_H be the amortized time cost of a hash table cache access. We define two key acceleration ratios:

- $\alpha_U = t_U/t_D$: the speedup of Direct Mapping over a raw utility call.
- $\alpha_{HT} = t_H/t_D$: the speedup of Direct Mapping over the hash table.

Given that a raw utility call (e.g., model training) is much more expensive than a cache lookup, we have $\alpha_U \gg 1$.

The total time for utility function evaluations in a Monte Carlo run is $n_p \times n \times t_U$ without any optimization. With our caching data structure, the expected total time becomes

$$\left(n_p \times n - 2 \sum_{l=0}^L n_{\text{dup}}(n_p, l) \right) \times t_U + n_p \times 2L \times t_D,$$

where the time saved is

$$\begin{aligned} t_{\text{sav}}(L) &= 2t_U \sum_{l=0}^L n_{\text{dup}}(n_p, l) - 2Ln_p t_D \\ &= 2t_D \left(\alpha_U \sum_{l=0}^L \left(n_p - n_l + n_l \left(1 - \frac{1}{n_l} \right)^{n_p} \right) - Ln_p \right) \end{aligned}$$

for a given storing range L .

Consequently, the optimal cache limit L^* is the value that maximizes the objective

$$L^* = \arg \max_{0 \leq L \leq n/2} t_{\text{sav}}(L). \quad (7)$$

Since $n_{\text{dup}}(n_p, l)$ decreases rapidly with l , the marginal benefit of expanding the cache also decreases. At the same time, a larger L increases memory allocation, initialization, and indexing overheads because more coalition sizes are included in the contiguous cache. As a result, the best runtime is usually achieved at a moderate cache limit rather than the largest feasible one. In our experiments, effective choices are typically small (e.g., $L = 4$ or 5 for $n = 30$), as discussed in Section 5.

4.4 Dense Indexing

The Coalition Size Narrowing strategy (Section 4.3) successfully confines our caching focus to the most beneficial coalition sizes ($|S| \leq L$ and $|S| \geq n - L$). However, a naive Direct Mapping still requires allocating a memory block of size 2^n to maintain offset integrity, rendering it prohibitively expensive.

To resolve this bottleneck, we propose a **Dense Indexing** scheme that compacts the sparse set of target coalitions into a contiguous memory block. Our goal is to construct a bijective map from $\{S \subseteq N : |S| \leq L\}$ to the integer interval $[0, K - 1]$, where $K = \sum_{l=0}^L \binom{n}{l}$.

The Mapping Function. We present $M_{\text{dense}}(S)$, a bijective mapping that achieves this goal. Our approach leverages the Combinatorial Number System (also known as combinadics) [18, 19], a classical framework that provides a canonical representation of a combination by its position in a lexicographically ordered list. To adapt this system for our variable-size caching requirement, we construct the mapping in two parts:

- (1) **Size-first ordering.** All coalitions of size l are ordered before any coalition of size $l + 1$.
- (2) **Lexicographic ordering within a size.** For coalitions of the same size, we use a standard lexicographic order based on player indices.

Given the grand player coalition set is $N = \{p_1, p_2, \dots, p_n\}$. For any coalition $S \subseteq N$ with $|S| = k$, let $S = \{p_{s_1}, p_{s_2}, \dots, p_{s_k}\}$ where $1 \leq s_1 < s_2 < \dots < s_k \leq n$. The dense index is defined as

$$M_{\text{dense}}(S) = \underbrace{\sum_{l=0}^{k-1} \binom{n}{l}}_{\text{Offset for smaller sizes}} + \underbrace{\sum_{t=1}^k \binom{s_t - 1}{t}}_{\text{Lexicographic rank within size } k}. \quad (8)$$

The first term accounts for all coalitions of size less than k . The second term computes the number of k -subsets that precede S in lexicographic order, using the standard combinatorics formula.

This construction guarantees that M_{dense} is a one-to-one mapping from the set $\{S \subseteq N : |S| \leq L\}$ to the integer interval $[0, \sum_{l=0}^L \binom{n}{l} - 1]$, thereby eliminating any memory waste.

Consider a scenario with $n = 5$ and a cache limit $L = 2$ to illustrate the sparsity problem (Table 2). Under Naive Mapping, the lexicographically last cached coalition $\{p_4, p_5\}$ maps to index $2^3 + 2^4 = 24$, necessitating a memory allocation of 25 slots. This

Table 2: Comparison for $n = 5, L = 2$. Naive Mapping spans indices 0–24 with significant gaps (e.g., index 7 of coalition $\{p_1, p_2, p_3\}$ is unused), whereas Dense Indexing perfectly packs all 16 valid coalitions into $[0, 15]$.

Size	Coalition S	M_{naive}	M_{dense}
Size 0	$\emptyset$	0 (00000 ₂)	0
Size 1	$\{p_1\}$	1 (00001 ₂)	1 ($\binom{5}{0} + \binom{0}{1}$)
	$\{p_2\}$	2 (00010 ₂)	2 ($\binom{5}{0} + \binom{1}{1}$)
	$\{p_3\}$	4 (00100 ₂)	3 ($\binom{5}{0} + \binom{2}{1}$)
	$\{p_4\}$	8 (01000 ₂)	4 ($\binom{5}{0} + \binom{3}{1}$)
	$\{p_5\}$	16 (10000 ₂)	5 ($\binom{5}{0} + \binom{4}{1}$)
Size 2	$\{p_1, p_2\}$	3 (00011 ₂)	6 ($\binom{5}{0} + \binom{2}{1} + \binom{0}{2} + \binom{1}{2}$)
	$\{p_1, p_3\}$	5 (00101 ₂)	7 ($\binom{5}{0} + \binom{2}{1} + \binom{0}{2} + \binom{2}{2}$)
	...	...	...
	$\{p_2, p_5\}$	18 (10010 ₂)	13 ($\binom{5}{0} + \binom{2}{1} + \binom{1}{2} + \binom{4}{2}$)
	$\{p_3, p_5\}$	20 (10100 ₂)	14 ($\binom{5}{0} + \binom{2}{1} + \binom{2}{2} + \binom{4}{2}$)
	$\{p_4, p_5\}$	24 (11000 ₂)	15 ($\binom{5}{0} + \binom{2}{1} + \binom{3}{2} + \binom{4}{2}$)
Memory Range Required		0 – 24 (Size 25)	0 – 15 (Size 16)

results in significant fragmentation, as indices corresponding to excluded size-3 coalitions (e.g., index 7 for $\{p_1, p_2, p_3\}$) remain unused, wasting $25 - 16 = 9$ slots. In contrast, Dense Indexing sequentially maps the $\binom{5}{0} + \binom{5}{1} + \binom{5}{2} = 16$ valid coalitions to the contiguous range $[0, 15]$. Crucially, as n increases, the divergence between the naive requirement (2^n) and the dense packing grows exponentially, rendering dense indexing essential for feasibility in high-dimensional spaces.

Correctness and Bijectivity. The validity of this compact indexing relies on the Combinatorial Number System [19], which establishes a canonical bijection between k -subsets of $\{1, \dots, n\}$ and integers in $[0, \binom{n}{k} - 1]$. Specifically, for a sorted coalition $S = \{p_{s_1}, \dots, p_{s_k}\}$ with $1 \leq s_1 < \dots < s_k \leq n$, the term $\sum_{t=1}^k \binom{s_t-1}{t}$ computes the exact number of k -subsets that are lexicographically smaller than S . Combined with the size-based offset $\sum_{l=0}^{k-1} \binom{n}{l}$, the mapping M_{dense} induces a total order over all coalitions of size at most L , guaranteeing a gap-free, bijective mapping to the interval $[0, \sum_{l=0}^L \binom{n}{l} - 1]$.

Efficient Implementation via Precomputation. A potential concern is the computational cost of evaluating Equation 8, which involves multiple combinatorial calculations. Evaluating factorials on the fly would be computationally expensive, potentially negating the benefits of caching. To address this, we implement a *Pascal’s Triangle Lookup Table* during the system initialization. We precompute and store $\binom{n}{k}$ values in a small 2D array. Consequently, computing $M_{\text{dense}}(S)$ requires only $|S|$ integer lookups and additions, which can be implemented efficiently with simple table lookups and additions. This ensures that the mapping overhead is negligible compared to the cost of a utility function evaluation. The time complexity of computing $M_{\text{dense}}(S)$ is $O(|S|)$ (assuming coalitions are sorted lists), which is same as $M_{\text{naive}}(S)$ and theoretically comparable to the cost of a hash table, but comparable in asymptotic cost to hash-table lookup, while avoiding hash computation and collision handling in the cached regions.

Handling Large Coalitions. To cache both small and large coalitions, we leverage the symmetry of the Shapley framework. For a coalition S with $|S| \geq n - L$, we instead cache its complement $N \setminus S$, which has size $n - |S| \leq L$. We store the pair $(M_{\text{dense}}(N \setminus S), U(S))$ in a second, separate memory chunk. This allows us to cache all coalitions of interest: both $|S| \leq L$ and $|S| \geq n - L$, using a total of only $2 \sum_{l=0}^L \binom{n}{l}$ memory locations. This represents a dramatic reduction from the naive $O(2^n)$ requirement to a manageable $O(\sum_{l=0}^L \binom{n}{l})$, making Direct Mapping practical for the Shapley computation scenarios considered in this paper.

4.5 Analysis of Caching Alternatives

Hybrid Caching. The analysis in Section 4.3 establishes that Direct Mapping is well suited to the high-density regimes at the extremes of the coalition spectrum. However, intermediate-sized coalitions ($L < |S| < n - L$), while sampled sparsely, still incur non-zero redundancy. This prompts a natural question: can we extract further efficiency by covering these sparse regions with a more flexible structure such as a hash table?

To investigate this potential, we model a **Hybrid Caching** architecture. We define two boundaries: L_D for Direct Mapping and L_H for the hash table ($L_D < L_H \leq n/2$). The strategy operates as follows:

- **Direct Mapping Region** ($|S| \leq L_D$ or $n - |S| \leq L_D$). Handled by M_{dense} for direct memory access.
- **Hash Table Region** ($L_D < |S| \leq L_H$ or $L_D < n - |S| \leq L_H$). Coalitions in this range are stored in a hash table to capture sporadic duplicates.
- **Uncached Region.** Coalitions outside these ranges are recomputed.

Given a fixed Direct Mapping limit L_D , the choice of the hash table limit L_H ($L_H > L_D$) primarily influences the time savings of the hybrid strategy, while its impact on total memory consumption is comparatively minor. The memory footprint is dominated by the Direct Mapping component, which scales with $\sum_{l=0}^{L_D} \binom{n}{l}$, whereas the hash table’s memory usage is proportional to the number of distinct intermediate-sized coalitions actually sampled, $n_p(L_H - L_D)$.

We formulate the time savings of this hybrid strategy, $t_{\text{sav}}(L_D, L_H)$, relative to the uncached baseline. Recalling that $\alpha_U = t_U/t_D$ and $\alpha_{HT} = t_H/t_D$, the savings can be expressed as

$$t_{\text{sav}}(L_D, L_H) = 2t_D \left[\underbrace{\alpha_U \sum_{l=0}^{L_H} n_{\text{dup}}(n_p, l)}_{\text{Saved Computation}} - \underbrace{L_D n_p}_{\text{Direct Overhead}} - \underbrace{\alpha_{HT}(L_H - L_D) n_p}_{\text{Hash Overhead}} \right].$$

This equation reveals the fundamental trade-off: extending coverage from L_D to L_H captures more duplicates (the first term) but incurs higher per-access overhead ($\alpha_{HT} > 1$) for every sample in that range.

To determine the optimal upper bound L_H , we analyze the marginal utility of expanding the hash table’s coverage. The incremental time saving Δt_{sav} gained by extending coverage from $L_H - 1$ to L_H is

$$\Delta t_{\text{sav}} = 2t_D (\alpha_U \cdot n_{\text{dup}}(n_p, L_H) - \alpha_{HT} \cdot n_p).$$

This gain remains positive only if the expected redundancy at size L_H exceeds a cost-ratio threshold

$$n_{\text{dup}}(n_p, L_H) \geq \frac{\alpha_{HT}}{\alpha_U} n_p. \quad (9)$$

Since we have established that $n_{\text{dup}}(n_p, l)$ is a non-increasing function of l while $l \leq n/2$, there exists a maximum L_H^* such that the above inequality holds for all $l \leq L_H^*$. This L_H^* gives the cutoff predicted by our cost model for the hash-table region, beyond which the cost of managing additional hash entries outweighs the benefit of the few extra cache hits.

However, this theoretical optimum hinges on an idealized model of hash table performance. In practice, the constant factors hidden in α_{HT} , including hash computation latency and memory allocation for dynamic entries, can be substantial. As we will demonstrate in Section 5, these real-world overheads often mean that the practical gains from caching intermediate-sized coalitions are marginal. In many scenarios, the hybrid strategy offers no advantage over a well-tuned Direct Mapping approach that focuses solely on the high-return extremes of the coalition size spectrum. The next section presents experiments that evaluate this design choice.

Adaptive Caching. A related alternative is adaptive caching, which monitors cache hit rates at runtime and disables caching for coalition sizes with low observed reuse. We do not use this strategy for two reasons. First, the reuse probability in Shapley approximation is largely determined by the coalition size and the sampling budget, which can be estimated before execution using our cost model. Second, runtime monitoring adds extra branches and state updates to the lookup path, which may offset the benefit of avoiding low-yield cache accesses. We empirically compare against an adaptive hash-table variant in Section 5.2, and the results show that it does not improve over the static Direct Mapping strategy in our evaluated settings.

5 EXPERIMENTS

In this section, we conduct a comprehensive experimental evaluation of CASH to evaluate its performance and applicability across diverse Shapley value computation tasks.

5.1 Cross-Algorithm & Cross-Dataset Generality

To evaluate the generality of CASH across algorithms and datasets, we conduct a comprehensive evaluation across six diverse UCI datasets (breast [38], car [5], heart [2], iris [1], statlog [13], wine [35]) and seven Shapley value approximation algorithms (Monte Carlo, Antithetic Sampling, Stratified Sampling, Complementary, SVARM, SVARM Stratified, GELS-Shapley). We fix the player (data provider) coalition size to $n = 30$ and set the cache limit to $L = 5$. Notably, we restrict the number of utility function evaluations to 20,000, which is a relatively conservative budget. This setup allows us to isolate the impact of data distribution and the algorithm’s sampling strategy on the caching performance.

The results, summarized in Figure 2, reveal two critical insights. First, CASH provides consistent speedups in our evaluated configurations: it delivers a positive speedup on all 42 (6 datasets $\times$ 7 algorithms) test configurations, with no performance degradation observed in these configurations. This robustness confirms

that CASH is a broadly applicable caching layer for the evaluated approximation methods.

Second, the magnitude of the speedup is primarily dictated by the algorithm’s sampling pattern, not the dataset. The high-return algorithms: GELS-Shapley, SVARM, and Stratified Sampling, consistently achieve speedups up to 29% across all datasets. This consistency is because their sampling strategies heavily concentrate on the small and large coalitions that our Direct Mapping cache is optimized for. In contrast, algorithms like Monte Carlo, Complementary, SVARM Stratified, and Antithetic Sampling, which either sample uniformly or actively promote diversity, exhibit a stable but more modest speedup of 8% to 11%, again with minimal variance across datasets.

These results suggest that the performance gain from the redundancy in the sampling process exists regardless of the dataset. This makes CASH a reliable and predictable optimization for the evaluated moderate-scale Shapley value computation tasks ($n \approx 30$).

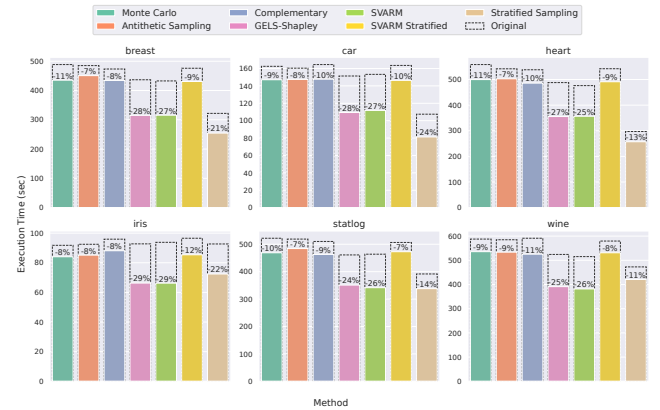


Figure 2: End-to-end runtime of seven Shapley value approximation algorithms across six UCI datasets ($n = 30$, 20,000 utility calls, $L = 5$). CASH consistently accelerates all methods, with speedups ranging from 8–11% for Monte Carlo variants to 24–29% for GELS-Shapley and SVARM.

5.2 Direct Mapping vs. Hash Table

To quantify the efficiency of different caching strategies, we benchmark three costs: (i) t_U , the cost of a raw utility evaluation; (ii) t_D , the cost of cache retrieval and storage using Direct Mapping; and (iii) t_H , the cost of cache retrieval and storage using a hash table. For both t_D and t_H , we include initialization overheads such as memory pre-allocation, so the reported numbers reflect the amortized cost within a full Shapley computation session. We report three ratios: $\alpha_U = t_U/t_D$, the speedup of Direct Mapping over raw execution; $\alpha_{HT} = t_H/t_D$, the relative speedup of Direct Mapping over the hash table; and $\alpha_U/\alpha_{HT} = t_U/t_H$, the speedup of the hash table over raw execution.

We use an MLP classifier as the utility function to provide a representative computational baseline. This is a conservative setting for many data analytics workloads, where $U(S)$ may involve more expensive operations such as complex query processing or deep

model training. Since cache lookup latency is independent of the utility function cost, a more expensive $U(S)$ would only increase the relative benefit of caching.

We fix the grand coalition size to $n = 30$ and vary the total number of utility evaluations from 1,500 to 150,000, corresponding to 50 to 5,000 permutations. For each method, we test cache limits $L \in \{1, 2, \dots, 8\}$, caching coalitions with $|S| \leq L$ or $|S| \geq n - L$.

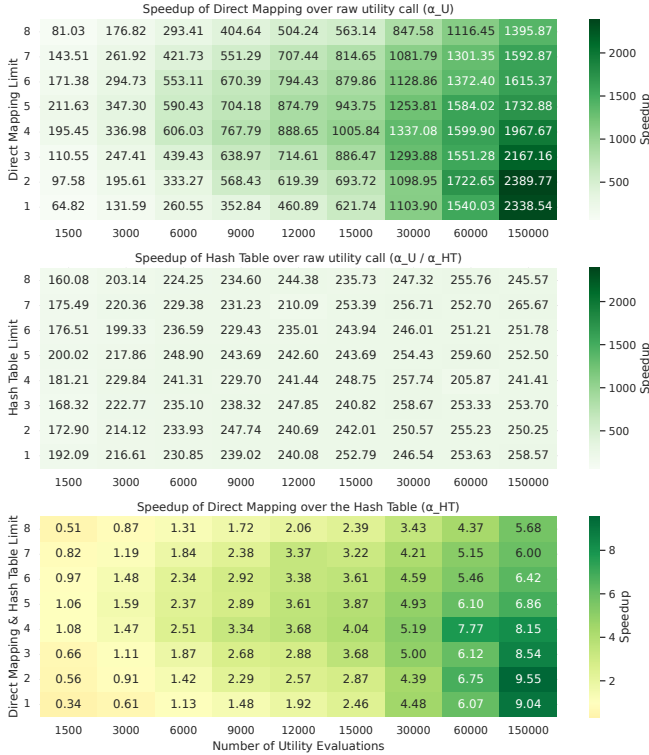


Figure 3: Results for $n = 30$. Top: the speedup of Direct Mapping over a raw utility call (α_U). Middle: the speedup of the hash table over a raw utility call (α_U / α_{HT}). Bottom: the relative speedup of Direct Mapping over the hash table (α_{HT}).

Figure 3 shows two main trends. First, the hash table provides stable but moderate acceleration, with a speedup of roughly 160–270 $\times$ over raw utility evaluation across most configurations. Second, Direct Mapping has a larger initialization cost, but becomes faster as the number of utility calls increases. For small workloads, such as 1,500 calls, Direct Mapping can be slower than the hash table because its pre-allocation cost is not yet amortized. However, for practical cache limits ($3 \leq L \leq 7$), Direct Mapping surpasses the hash table at around 3,000 utility evaluations and reaches 1300–2400 $\times$ speedup over raw utility evaluation at 150,000 calls. These results support our design choice: hash tables are a useful general memoization baseline, but Direct Mapping is better suited to high-throughput Shapley value approximation workloads.

C++ Microbenchmark. To check whether the advantage of Direct Mapping is specific to Python dictionaries, we further implement a C++ lookup microbenchmark compiled with `-O3 -march=native`.

This benchmark removes Python interpreter overhead and compares Direct Mapping with a pre-reserved `std::unordered_map`. We also include an adaptive hash table variant that disables caching for coalition sizes whose observed hit rate falls below 1%. The experiment runs on a server with AMD EPYC 9754 and DDR5 memory.

Table 3: C++ microbenchmark: total lookup time in microseconds for selected configurations at $n = 30$. Direct Mapping remains faster than hash-based alternatives after removing Python overhead.

Method	15,000 queries			150,000 queries		
	$L = 4$	$L = 5$	$L = 6$	$L = 4$	$L = 5$	$L = 6$
Static Hash Table	1008	2186	6606	7179	8529	16398
Adaptive Hash Table	889	2785	11151	4224	6417	15463
CASH Direct Mapping	346	777	2714	2521	3229	5978
Static / Direct	2.91 $\times$	2.81 $\times$	2.43 $\times$	2.85 $\times$	2.64 $\times$	2.74 $\times$

Table 3 shows that Direct Mapping remains faster than hash-based caching even in a low-level implementation. Across the selected configurations, it achieves a 2.4–2.9 $\times$ speedup over the static hash table. The adaptive hash table does not consistently improve performance. Although it can reduce some unnecessary lookups, its runtime monitoring adds branches and state updates to the lookup path. These results indicate that the benefit of Direct Mapping is not solely due to Python overhead, although the exact speedup ratio depends on the implementation language.

Table 4: End-to-end runtime with utility functions of different costs ($n = 30$, $L = 5$). CASH provides consistent relative time reduction across models, while the absolute saving increases for more expensive utilities.

Utility Model $U(S)$	n_p	Uncached (s)	CASH (s)	Time Saved
Logistic Regression ($\alpha_U \approx 1800$)	10	4.98	4.72	5.22%
	100	50.99	45.47	10.83%
	1,000	515.78	441.03	14.49%
	10,000	5,095.22	3,951.81	22.44%
SVM (RBF Kernel) ($\alpha_U \approx 160$)	10	0.65	0.62	4.62%
	100	6.55	5.81	11.28%
	1,000	65.38	56.16	14.10%
	10,000	650.29	511.26	21.38%
MLP Neural Network ($\alpha_U \approx 56000$)	10	455.67	430.84	5.45%
	100	4,737.82	4,135.09	12.72%
	1,000	47,945.09	40,685.63	15.14%
	10,000	468,119.64	366,850.91	21.63%

5.3 Impact of Utility Function Cost

In practical data valuation pipelines, the utility function $U(S)$ often involves model training or inference rather than a lightweight arithmetic operation. We therefore evaluate whether the lookup overhead of CASH remains negligible when $U(S)$ is instantiated with machine learning workloads of different costs. We consider three utility models: Logistic Regression, an RBF-kernel SVM, and a Multi-Layer Perceptron (MLP). We fix $n = 30$, use Monte Carlo

permutation sampling, set the cache limit to $L = 5$, and vary the number of sampled permutations n_p from 10 to 10,000.

Table 4 shows that the relative time reduction mainly follows the redundancy level induced by the sampling budget. At $n_p = 1,000$, CASH reduces runtime by about 14%–15% across all three models, which is consistent with the expected duplicate rate of approximately 14.8% for this setting. When n_p increases to 10,000, the duplicate rate becomes higher, and the time reduction increases to around 21%–22%.

The similar relative savings across Logistic Regression, SVM, and MLP indicate that the lookup overhead of CASH is small compared with the cost of realistic utility evaluations. At the same time, the absolute saving grows with the cost of $U(S)$. For example, with the MLP utility and $n_p = 10,000$, CASH reduces the runtime from 468,119.64 seconds to 366,850.91 seconds, saving about 28.1 hours. These results support the use of CASH in data valuation workloads where repeated utility evaluations dominate the total runtime.

5.4 The (In)Effectiveness of the Hybrid Strategy

Our theoretical analysis in Section 4 suggests a hybrid caching architecture that combines the ultra-low latency of Direct Mapping for extreme-sized coalitions with the flexible coverage of a hash table for intermediate sizes. To empirically evaluate this design, we experiment on a coalition size of $n = 30$, systematically varying the Direct Mapping limit L_D and the hash table limit L_H (where $L_D \leq L_H \leq n/2$). The results are presented in Figure 4, where each cell shows the normalized runtime relative to the uncached baseline.

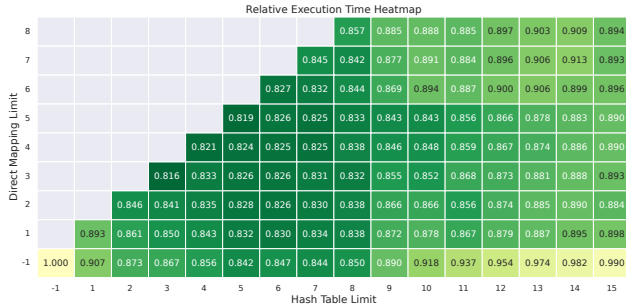


Figure 4: Normalized runtime of the hybrid caching strategy ($n = 30$) relative to the uncached baseline (lower values indicate better performance). The parameters L_D and L_H represent the size limits for Direct Mapping and the hash table. The diagonal ($L_D = L_H$) denotes pure Direct Mapping (which yields the lowest runtime), while the bottom row ($L_D = -1$) denotes pure hash table caching. The results show that hybrid configurations (off-diagonal) are less effective than pure Direct Mapping.

The data show two trends. First, **the best performance is consistently achieved on the diagonal**, where $L_D = L_H$. This configuration corresponds to using only Direct Mapping and disabling the hash table component entirely. For example, the global optimum is at ($L_D = 5, L_H = 5$) with a runtime of 81.9%, while any off-diagonal

point, such as ($L_D = 4, L_H = 5$), yields a slower runtime of 82.4%. This finding indicates that the overhead of managing the hash table, including dynamic memory allocation, hash computation, and cache misses, outweighs the modest gains from caching the rarely duplicated intermediate-sized coalitions.

Second, and perhaps more revealingly, **the pure hash table strategy (bottom row, $L_D = -1$) exhibits a clear performance degradation beyond a certain L_H** . As L_H increases past 7, the relative runtime begins to rise, demonstrating that blindly caching more coalitions is counterproductive. This happens because the hash table incurs significant management costs for every unique coalition it stores, but for intermediate sizes, the duplicate rate is so low that these costs are never amortized by cache hits. In essence, the hash table becomes a liability rather than an asset.

Consequently, our empirical evaluation demonstrates that for typical Shapley value computation workloads (e.g., $n \approx 30$), a well-tuned pure Direct Mapping strategy is simpler and performs better than the tested hybrid and pure-hash-table alternatives in our experiments. This insight greatly simplifies the deployment of CASH: practitioners need only select an optimal L for Direct Mapping, avoiding the complexity and pitfalls of managing a secondary, underperforming hash table. This result highlights the importance of aligning system design with the intrinsic structural properties of the problem: in this case, the skewed distribution of redundant evaluations.

5.5 Cache Limits and Player Scales

To evaluate the end-to-end effectiveness of our Direct Mapping strategy, we measure the total runtime of Monte Carlo Shapley value approximation with and without caching across a wide range of problem scales and cache configurations. Using the breast dataset, we vary the coalition size $n \in \{20, 30, 40, 50\}$ and, for each n , test cache limits $L \in \{1, \dots, L_{\max}\}$ (where $L_{\max}$ is 8 for $n = 20, 30, 40$ and 6 for $n = 50$ due to the memory limit). The total number of utility function evaluations is varied from 10,000 to 100,000. In all cases, we cache coalitions with size $|S| \leq L$ or $|S| \geq n - L$, with the baseline ($L = -1$) performing no caching.

Figure 5 reports the normalized runtime, i.e., the fraction of time taken relative to the uncached baseline. Our results reveal three key insights that emerge from this joint analysis of n and L :

(1) **Consistent and significant speedup.** Across all tested workloads, Direct Mapping reduces end-to-end computation time by 8–36%. This gain comes from reusing repeated coalition utilities rather than changing the estimator; no approximation or accuracy loss is involved, as shown in Figure 6.

(2) **Existence of an optimal cache limit L that depends on n .** For any fixed n , performance is not monotonic in L . Small L (e.g., $L = 1, 2$) provides limited coverage and modest speedup. Increasing L improves performance up to a point, but larger L incurs diminishing returns or even degradation at low call volumes due to memory allocation and indexing overhead. Crucially, the optimal L shifts with n : for smaller problems like $n = 20$, a larger L (e.g., $L = 8$) is beneficial, while for larger problems like $n = 50$, a more conservative L (e.g., $L = 2-3$) is preferred.

(3) **Speedup decays predictably with increasing n .** The maximum achievable speedup is highly sensitive to the coalition size

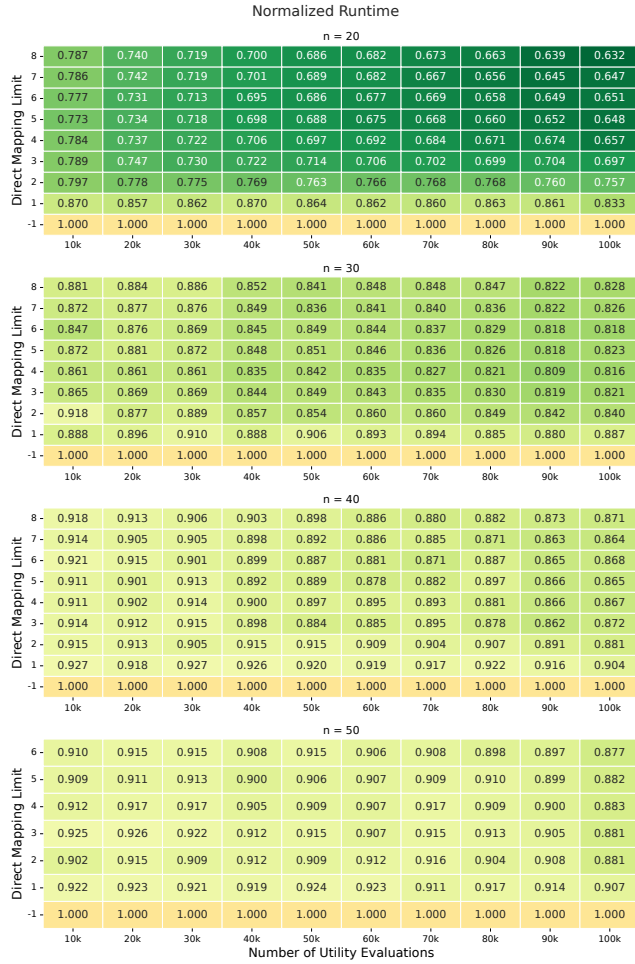


Figure 5: Normalized runtime (relative to the uncached baseline) of Monte Carlo Shapley value approximation for different coalition sizes n , and cache limits L . The results demonstrate that Direct Mapping provides consistent speedups, with the optimal L and maximum acceleration decreasing as n increases due to combinatorial explosion.

n . When $n = 20$, Direct Mapping reduces runtime by up to 36.8% at $L = 8$, due to the small combinatorial space and high cache hit rates. At $n = 30$, the best speedup drops to 17%, and at $n = 40$, it further declines to 14%. For $n = 50$, even with aggressive caching ($L = 6$), the improvement stabilizes around 12–13%.

This decay is driven by the growth of the coalition space. For a fixed coalition size l , the number of possible coalitions grows as $\binom{n}{l}$, so a fixed sampling budget covers a smaller fraction of the space as n increases. As a result, duplicate evaluations become less frequent, and the benefit of exact-match caching decreases. This observation is consistent with our analysis in Section 3, where the expected duplicate count $n_{\text{dup}}(n_p, l)$ decreases as $\binom{n}{l}$ increases.

Therefore, CASH is most suitable for moderate-scale settings where exact coalition reuse remains frequent. This regime arises in several Shapley-based analytics workloads. For example, in feature

attribution, players are input features, and many tabular SHAP tasks involve tens of features. In organization-level data valuation, including some federated or collaborative settings, players may correspond to clients, companies, hospitals, or data silos rather than individual records. We emphasize that when valuation is performed at a fine-grained record level, the number of players can be much larger and the benefit of exact-match caching is expected to decrease. In our experiments, small cache limits such as $L = 4$ or $L = 5$ work well for the common setting around $n = 30$.

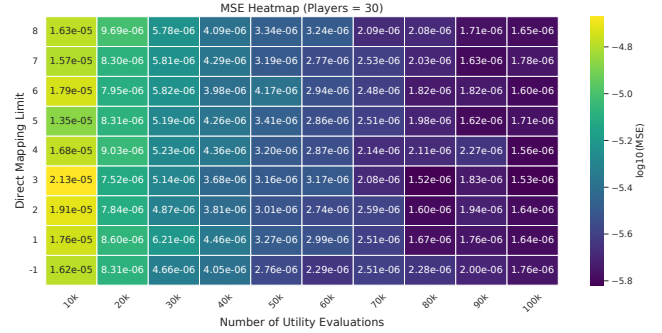


Figure 6: Mean Squared Error (MSE) of Monte Carlo Shapley value approximation for $n = 30$ across different cache limits L and numbers of utility function evaluations. The results confirm that CASH introduces no additional approximation error, as the MSE is nearly identical to the uncached baseline ($L = -1$) for all configurations.

5.6 Memory Footprint Analysis

While the primary goal of CASH is to reduce computation time, its memory overhead is important for deployment. Direct Mapping has a deterministic memory footprint that is independent of the sampling volume n_p . The memory block is pre-allocated according to $2 \sum_{l=0}^L \binom{n}{l}$, so increasing the number of sampled permutations does not increase cache memory usage.

Table 5: Peak memory usage (in MiB) of CASH. Memory grows with n and L , but the empirically effective configurations use modest memory.

Scale	Baseline	$L = 3$	$L = 4$	$L = 5$	$L = 6$	$L = 7$	$L = 8$
$n = 20$	0.69	0.70	0.71	0.71	0.71	1.08	2.04
$n = 30$	0.69	0.73	0.74	1.38	5.91	21.44	66.10
$n = 40$	0.69	0.77	0.86	5.88	35.17	177.41	764.14
$n = 50$	0.69	0.83	2.04	18.20	139.44	OOM	OOM

Note: Baseline denotes execution without caching. OOM = Out of Memory.

Table 5 reports the empirical peak memory usage of the Monte Carlo sampler with CASH. The results show a clear growth pattern as L increases. For example, at $n = 40$, increasing L from 4 to 8 increases memory usage from 0.86 MiB to 764.14 MiB. This confirms that aggressive caching of larger coalition sizes can be memory intensive.

However, the runtime results in Figure 5 show that larger L does not necessarily lead to better performance. Once the duplicate rate becomes low for intermediate-sized coalitions, the extra memory allocation and indexing overhead may outweigh the few additional cache hits. Thus, the memory-speedup tradeoff favors a conservative cache limit rather than caching as many coalitions as possible.

In the tested workloads, the empirically best configurations require modest memory. For example, peak acceleration for $n = 40$ is achieved at $L = 4$, requiring only 0.86 MiB, while the effective setting $L = 5$ for $n = 30$ uses 1.38 MiB. These results provide practical guidance for selecting L : users should choose a small cache limit that captures high-duplicate coalition sizes while avoiding the memory growth caused by larger L .

5.7 Scalability in Multi-threaded Environments

To evaluate the behavior of CASH under multi-threaded execution of our Direct Mapping architecture (Section 4.2), we evaluate the performance of CASH under varying degrees of parallelism. Using the breast dataset ($n = 30$) with a workload of 100,000 utility evaluations, we scale the number of execution threads from 2 to 32.

Table 6 details the runtime comparison. CASH consistently outperforms the uncached baseline across all concurrency levels. Crucially, even at 32 threads, CASH maintains a time reduction of 12.2% (from 106.6s to 93.6s). The marginal decrease in relative gain (from 16.4% at 2 threads) may be related to hardware-level resource contention at high concurrency. These results suggest that CASH can retain useful speedups under multi-threaded execution.

Table 6: Multi-threaded performance comparison ($n = 30$, 100k calls). CASH maintains consistent speedups without lock contention even at high concurrency.

Threads	2	4	8	16	32
Baseline (sec)	1209.7	626.6	320.0	160.2	106.6
CASH (sec)	1011.3	536.2	274.8	138.2	93.6
Time Reduction	16.4%	14.4%	14.1%	13.7%	12.2%

5.8 Accelerating SHAP

To evaluate the applicability of CASH to model interpretability, we test it on the Adult Income dataset with $n = 14$ features and 325 instances, using a K-neighbors classifier. To isolate the effect of caching, we compare two implementations with the same underlying Monte Carlo permutation sampler: (1) a pure Python Monte Carlo sampler without caching, and (2) the same sampler augmented with CASH ($L = 7$). We also report the runtime of the official `shap.PermutationExplainer` as an external reference, rather than as a direct algorithmic baseline.

Figure 7 shows that adding CASH reduces the runtime of the Monte Carlo sampler from 8.83 seconds to 3.98 seconds, corresponding to a 2.22 $\times$ speedup. Since the two implementations use the same sampling procedure, this improvement comes from reusing repeated coalition evaluations. The MSE curves are also nearly identical, converging to around 3.3×10^{-5} , which confirms that caching does not change the approximation result.

The `shap.PermutationExplainer` takes 5.62 seconds to reach a similar MSE of around 3.4×10^{-5} in this setting. This comparison is not intended to isolate caching alone, since the official implementation includes its own engineering choices. Instead, it shows that a simple Monte Carlo implementation equipped with CASH can be competitive with a widely used SHAP implementation on compact feature spaces. These results suggest that CASH is useful for SHAP-style interpretability tasks where the number of features is moderate and repeated feature subsets occur frequently.

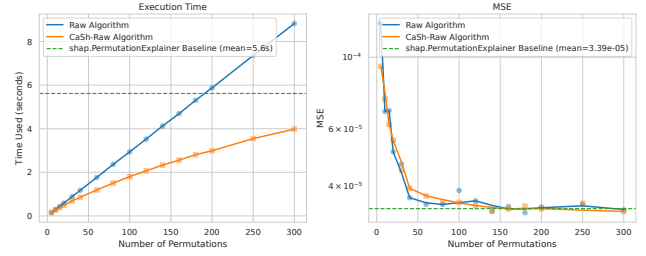


Figure 7: Performance comparison of SHAP implementations on the Adult Income dataset ($n = 14$ features).

6 CONCLUSION

In this paper, we propose CASH, an algorithm-agnostic caching framework that accelerates sampling-based Shapley value approximation via Direct Mapping. Our analysis confirms that sampling redundancy is structurally skewed toward extreme coalition sizes, justifying our size-aware memory optimization. Extensive experiments demonstrate that CASH reduces computation time by 8% to 29% across diverse settings, with peak gains in strategies that heavily sample small and large coalitions, without introducing additional approximation error beyond the underlying estimator. Furthermore, we verify that our C++ microbenchmark suggests that the benefit of Direct Mapping is not solely due to Python overhead, and show that CASH can provide substantial absolute time savings for expensive utility functions and useful speedups for SHAP-style interpretability tasks.

For future work, we will explore two directions. First, to scale beyond moderate player sets ($n > 50$), we aim to investigate approximate cache reuse. Retrieving similar cached coalitions as warm starts or control variates may offer a way to explore new bias-variance trade-offs. Second, we plan to extend CASH to distributed environments. Because deterministic utility queries yield immutable cache entries, the deterministic nature of utility queries may allow lightweight parallel caching designs, which motivates future study of scalable distributed caching architectures.

ACKNOWLEDGMENTS

This work was supported in part by NSFC (U23A20306, 62472378, U25A20430), the Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars (LR25F020001), and the Zhejiang Province Pioneer Plan (2024C01074, 2025C01084).

REFERENCES

- [1] Fisher R. A. 1936. Iris. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C56C76>.
- [2] Janosi Andras, Steinbrunn William, Pfisterer Matthias, and Detrano Robert. 1989. Heart Disease. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C52P4X>.
- [3] Leopoldo Bertossi, Benny Kimelfeld, Ester Livshits, and Mikaël Monet. 2023. The Shapley Value in Database Management. *SIGMOD Rec.* 52, 2 (Aug. 2023), 6–17. <https://doi.org/10.1145/3615952.3615954>
- [4] Yuran Bi, Jinfei Liu, Chen Zhao, Junyi Zhao, Kui Ren, and Li Xiong. 2024. Share: Stackelberg-Nash based Data Markets. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 3573–3586. <https://doi.org/10.1109/ICDE60146.2024.00275>
- [5] Marko Bohanec. 1988. Car Evaluation. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5JP48>.
- [6] Javier Castro, Daniel Gómez, Elisenda Molina, and Juan Tejada. 2017. Improving polynomial estimation of the Shapley value by stratified random sampling with optimum allocation. *Computers & Operations Research* 82 (2017), 180–188. <https://doi.org/10.1016/j.cor.2017.01.019>
- [7] Susan Davidson, Daniel Deutch, Nave Frost, Benny Kimelfeld, Omer Koren, and Mikaël Monet. 2022. ShapGraph: An Holistic View of Explanations through Provenance Graphs and Shapley Values. In *Proceedings of the 2022 International Conference on Management of Data (Philadelphia, PA, USA) (SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2373–2376. <https://doi.org/10.1145/3514221.3520172>
- [8] Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. 2022. Computing the Shapley Value of Facts in Query Answering. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 1570–1583. <https://doi.org/10.1145/3514221.3517912>
- [9] Zhenan Fan, Huang Fang, Xinglu Wang, Zirui Zhou, Jian Pei, Michael P. Friedlander, and Yong Zhang. 2025. Fair and efficient contribution valuation for vertical federated learning. arXiv:2201.02658 [cs.LG] <https://arxiv.org/abs/2201.02658>
- [10] Eitan Farchi, Ramasuri Narayanam, and Lokesh Nagalapatti. 2021. Ranking Data Slices for ML Model Validation: A Shapley Value Approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 1937–1942. <https://doi.org/10.1109/ICDE51399.2021.00180>
- [11] Amirata Ghorbani, Michael P. Kim, and James Zou. 2020. A Distributional Framework For Data Valuation. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research)*, Vol. 119. PMLR, 3535–3544. <http://proceedings.mlr.press/v119/ghorbani20a.html>
- [12] Amirata Ghorbani and James Y. Zou. 2019. Data Shapley: Equitable Valuation of Data for Machine Learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 2242–2251. <http://proceedings.mlr.press/v97/ghorbani19c.html>
- [13] Hofmann Hans. 1994. Statlog (German Credit Data). UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5NC77>.
- [14] Robert Hu, Siu Lun Chau, Jaime Ferrando Huertas, and Dino Sejdinovic. 2022. Explaining Preferences with Shapley Values. arXiv:2205.13662 [stat.ML] <https://arxiv.org/abs/2205.13662>
- [15] Hoang Anh Just, Feiyang Kang, Jiachen T. Wang, Yi Zeng, Myeongseob Ko, Ming Jin, and Ruoxi Jia. 2023. LAVA: Data Valuation without Pre-Specified Learning Algorithms. arXiv:2305.00054 [cs.LG] <https://arxiv.org/abs/2305.00054>
- [16] Bojan Karlas, David Dao, Matteo Interlandi, Bo Li, Sebastian Schelter, Wentao Wu, and Ce Zhang. 2022. Data Debugging with Shapley Importance over End-to-End Machine Learning Pipelines. *CoRR* abs/2204.11131 (2022). <https://doi.org/10.48550/ARXIV.2204.11131> arXiv:2204.11131
- [17] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. [n.d.]. The UCI Machine Learning Repository. <https://archive.ics.uci.edu>
- [18] Gary D. Knott. 1974. A numbering system for combinations. *Commun. ACM* 17, 1 (Jan. 1974), 45–46. <https://doi.org/10.1145/360767.360811>
- [19] Donald E Knuth. 2011. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional.
- [20] Patrick Kolpaczki, Viktor Bengs, Maximilian Muschalik, and Eyke Hüllermeier. 2024. Approximating the Shapley Value without Marginal Contributions. arXiv:2302.00736 [cs.LG] <https://arxiv.org/abs/2302.00736>
- [21] Yongchan Kwon and James Zou. 2022. Beta Shapley: a Unified and Noise-reduced Data Valuation Framework for Machine Learning. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event (Proceedings of Machine Learning Research)*, Gustavo Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera (Eds.), Vol. 151. PMLR, 8780–8802. <https://proceedings.mlr.press/v151/kwon22a.html>
- [22] Weida Li and Yaoliang Yu. 2024. Faster Approximation of Probabilistic and Distributional Values via Least Squares. In *The Twelfth International Conference on Learning Representations*. <https://openreview.net/forum?id=IvSMIsztkA>
- [23] Hong Lin, Shixin Wan, Zhongle Xie, Ke Chen, Meihui Zhang, Lidan Shou, and Gang Chen. 2025. A Comprehensive Study of Shapley Value in Data Analytics. *Proc. VLDB Endow.* 18, 9 (2025), 3077–3092. <https://doi.org/10.14778/3746405.3746429>
- [24] Jinfei Liu, Jian Lou, Junxu Liu, Li Xiong, Jian Pei, and Jimeng Sun. 2021. Dealer: An End-to-End Model Marketplace with Differential Privacy. *Proc. VLDB Endow.* 14, 6 (2021), 957–969. <https://doi.org/10.14778/3447689.3447700>
- [25] Ester Livshits, Leopoldo Bertossi, Benny Kimelfeld, and Moshe Sebag. 2021. The Shapley Value of Tuples in Query Answering. *Logical Methods in Computer Science* Volume 17, Issue 3 (Sept. 2021). [https://doi.org/10.46298/lmcs-17\(3:22\)2021](https://doi.org/10.46298/lmcs-17(3:22)2021)
- [26] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 4765–4774. <https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html>
- [27] Xuan Luo and Jian Pei. 2024. Applications and Computation of the Shapley Value in Databases and Machine Learning. In *Companion of the 2024 International Conference on Management of Data, SIGMOD conference 2024 (Santiago AA, Chile) (SIGMOD '24)*. Association for Computing Machinery, New York, NY, USA, 630–635. <https://doi.org/10.1145/3626246.3654680>
- [28] Sasan Maleki, Long Tran-Thanh, Greg Hines, Talal Rahwan, and Alex Rogers. 2014. Bounding the Estimation Error of Sampling-based Shapley Value Approximation. arXiv:1306.4265 [cs.GT] <https://arxiv.org/abs/1306.4265>
- [29] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2011. The Complexity of Causality and Responsibility for Query Answers and non-Answers. arXiv:1009.2021 [cs.DB] <https://arxiv.org/abs/1009.2021>
- [30] Rory Mitchell, Joshua Cooper, Eibe Frank, and Geoffrey Holmes. 2022. Sampling Permutations for Shapley Value Estimation. arXiv:2104.12199 [stat.ML] <https://arxiv.org/abs/2104.12199>
- [31] Rory Mitchell, Joshua Cooper, Eibe Frank, and Geoffrey Holmes. 2022. Sampling Permutations for Shapley Value Estimation. *J. Mach. Learn. Res.* 23 (2022), 43:1–43:46. <https://jmlr.org/papers/v23/21-0439.html>
- [32] Jian Pei. 2022. A Survey on Data Pricing: From Economics to Data Science. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (Oct. 2022), 4586–4608. <https://doi.org/10.1109/tkde.2020.3045927>
- [33] Stephanie Schoch, Haifeng Xu, and Yangfeng Ji. 2022. CS-Shapley: Class-wise Shapley Values for Data Valuation in Classification. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/df334022279996b07e0870a629c18857-Abstract-Conference.html
- [34] Lloyd S. Shapley. 1951. *Notes on the N-Person Game – II: The Value of an N-Person Game*. RAND Corporation, Santa Monica, CA. <https://doi.org/10.7249/RM0670>
- [35] Aeberhard Stefan and Forina M. 1992. Wine. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5PC7J>.
- [36] Mukund Sundararajan and Amir Najmi. 2020. The many Shapley values for model explanation. arXiv:1908.08474 [cs.AI] <https://arxiv.org/abs/1908.08474>
- [37] Jiachen T. Wang, Tianji Yang, James Zou, Yongchan Kwon, and Ruoxi Jia. 2024. Rethinking Data Shapley for Data Selection Tasks: Misleads and Merits. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. <https://openreview.net/forum?id=mKYBMf1hHG>
- [38] William Wolberg, Olvi Mangasarian, Nick Street, and W. Street. 1993. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5DW2B>.
- [39] Haocheng Xia, Jiayao Zhang, Qiheng Sun, Jinfei Liu, Kui Ren, Li Xiong, and Jian Pei. 2025. Computing Shapley Values for Dynamic Data. *IEEE Transactions on Knowledge and Data Engineering* 37, 6 (2025), 3253–3271. <https://doi.org/10.1109/TKDE.2025.3527380>
- [40] Jiayao Zhang, Yuran Bi, Mengye Cheng, Jinfei Liu, Kui Ren, Qiheng Sun, Yihang Wu, Yang Cao, Raul Castro Fernandez, Haifeng Xu, Ruoxi Jia, Yongchan Kwon, Jian Pei, Jiachen T. Wang, Haocheng Xia, Li Xiong, Xiaohui Yu, and James Zou. 2024. A Survey on Data Markets. *CoRR* abs/2411.07267 (2024). <https://doi.org/10.48550/ARXIV.2411.07267> arXiv:2411.07267
- [41] Jiayao Zhang, Qiheng Sun, Jinfei Liu, Li Xiong, Jian Pei, and Kui Ren. 2023. Efficient Sampling Approaches to Shapley Value Approximation. *Proc. ACM Manag. Data* 1, 1, Article 48 (May 2023), 24 pages. <https://doi.org/10.1145/3588728>
- [42] Shuyuan Zheng, Yang Cao, and Masatoshi Yoshikawa. 2023. Secure Shapley Value for Cross-Silo Federated Learning. *Proceedings of the VLDB Endowment* 16, 7 (March 2023), 1657–1670. <https://doi.org/10.14778/3587136.3587141>
- [43] Yiding Zhu, Hongwei Zhang, Jiayao Zhang, Jinfei Liu, and Kui Ren. 2024. DataPrice: An Interactive System for Pricing Datasets in Data Marketplaces. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4433–4436. <https://doi.org/10.14778/3685800.3685893>