A Survey of Data Prefetching and Speculative Execution Techniques for Hardware Optimization

www.surveyx.cn

Abstract

Data prefetching and speculative execution are pivotal in enhancing hardware performance, addressing high memory latency, and optimizing processor throughput. This survey explores these techniques, emphasizing their integration in modern computing systems to exploit instruction-level and thread-level parallelism. Data prefetching, exemplified by strategies like the Multi-Lookahead Offset Prefetcher (MLOP), enhances cache efficiency and system performance, achieving significant speedups. Speculative execution, crucial in large language models (LLMs) and distributed systems, improves throughput but poses security risks, as seen in vulnerabilities like Spectre and Meltdown. The survey underscores the necessity for robust security frameworks to mitigate these threats. It highlights the importance of adaptive prefetching strategies and learning-based approaches to optimize cache performance, as well as innovative architectures that integrate static, dynamic, and speculative methods for hardware optimization. Despite advancements, challenges persist in accurately modeling speculative execution and evaluating cache performance under diverse workloads. The survey calls for future research to develop secure speculative mechanisms, explore adaptive algorithms, and integrate machine learning techniques to enhance hardware efficiency. Addressing these areas will ensure the advancement of hardware optimization techniques, balancing performance and security in modern computing environments.

1 Introduction

1.1 Significance of Data Prefetching and Speculative Execution

Data prefetching and speculative execution are essential techniques that enhance hardware performance in modern computing systems, particularly in addressing high memory access latency and the speed disparity between processors and memory. Architectures like Dynamic Simultaneous Multithreading (DSMT) utilize these techniques to exploit instruction-level parallelism (ILP) and thread-level parallelism (TLP), optimizing performance across various workloads [1]. In data-intensive applications, notably in serverless computing, these strategies are crucial for efficient data processing and resource utilization [2].

Data prefetching predicts and preloads necessary data into cache, thus mitigating memory latency and enhancing cache efficiency. Techniques such as the Multi-Lookahead Offset Prefetcher (MLOP) illustrate how prefetching can improve cache miss coverage and timeliness, leading to overall system performance gains [3]. Speculative execution allows processors to execute instructions ahead of time, increasing parallelism and throughput, which is particularly important in large language models (LLMs) where it helps reduce inference latency [4].

Despite its advantages, speculative execution introduces security vulnerabilities, as evidenced by attacks like Spectre and Meltdown, which exploit speculative execution to leak sensitive information through side channels [5]. Addressing these vulnerabilities is critical, and ongoing research is focused on developing secure speculative mechanisms [6]. Nevertheless, speculative execution remains a

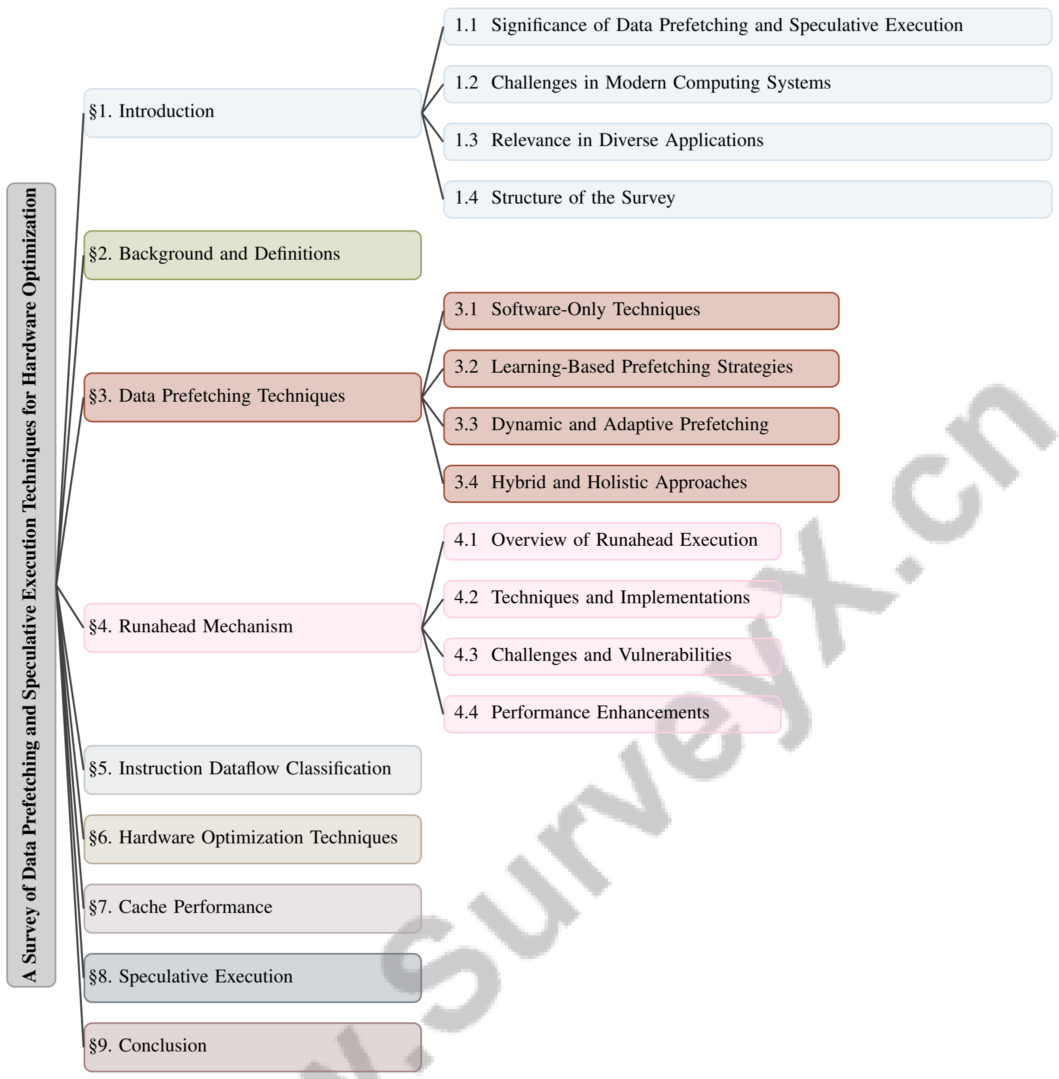

A Survey of Data Prefetching and Speculative Execution Techniques for Hardware Optimization
§1. Introduction
1.1 Significance of Data Prefetching and Speculative Execution
1.2 Challenges in Modern Computing Systems
1.3 Relevance in Diverse Applications
1.4 Structure of the Survey
§2. Background and Definitions
§3. Data Prefetching Techniques
3.1 Software-Only Techniques
3.2 Learning-Based Prefetching Strategies
3.3 Dynamic and Adaptive Prefetching
3.4 Hybrid and Holistic Approaches
§4. Runahead Mechanism
4.1 Overview of Runahead Execution
4.2 Techniques and Implementations
4.3 Challenges and Vulnerabilities
4.4 Performance Enhancements
§5. Instruction Dataflow Classification
§6. Hardware Optimization Techniques
§7. Cache Performance
§8. Speculative Execution
§9. Conclusion

Figure 1: chapter structure

vital component of modern microarchitectural features such as out-of-order execution, enhancing hardware efficiency [7].

Moreover, speculative execution optimizes concurrency in distributed systems, such as Ethereum smart contracts, which face challenges from the need for sequential execution despite multicore architecture availability [8]. By integrating data prefetching and speculative execution, modern computing systems can achieve significant performance improvements, effectively addressing high memory access latency in big-data workloads and optimizing interactive data visualization and cloud processing systems.

1.2 Challenges in Modern Computing Systems

Modern computing systems face numerous challenges that data prefetching and speculative execution techniques aim to address. A primary issue is the inefficiency of existing prefetchers, which often fail to accurately identify data streams, resulting in delayed selection processes and suboptimal performance [9]. This inefficiency is especially evident in cloud workloads, where the lack of dominant strides diminishes the effectiveness of traditional prefetching strategies [10]. The unpredictable and non-linear behavior of I/O workloads further complicates the efficacy of conventional caching algorithms like LRU, which struggle to adapt to dynamic conditions [11].

Additionally, the absence of prefetch or read-ahead functions in current memory object caching systems significantly hampers real-time data processing capabilities [12]. Inaccurate wrong-path prefetches and untimely prefetches contribute to cache pollution and inefficiencies in instruction fetching [13]. The dynamic behavior of threads and frequent misspeculation also lead to diminishing performance returns, highlighting the need for more adaptive prefetching strategies [1].

Security vulnerabilities associated with speculative execution, including Meltdown, Spectre, and L1 terminal fault (L1TF) attacks, pose significant challenges [14]. Existing kernel safety mechanisms, particularly layout randomization, inadequately protect against speculative execution and side-channel attacks [5]. The lack of systematic modeling of these attacks complicates the understanding of their characteristics and the evaluation of existing defenses [15]. Furthermore, slow detection of speculative information leaks in black-box CPUs limits thorough testing and the identification of new vulnerability classes [16].

Reliance on external services for data passing in serverless workflows introduces significant latency, particularly during cold starts, which data prefetching and speculative execution aim to alleviate [2]. Additionally, the sequential execution of Ethereum smart contracts leads to inefficiencies as transaction volumes rise, necessitating benchmarks to evaluate speculative execution strategies [8]. Addressing these multifaceted challenges is essential for enhancing the performance, security, and efficiency of modern computing environments, necessitating ongoing innovation in data prefetching and speculative execution strategies.

1.3 Relevance in Diverse Applications

Data prefetching and speculative execution are critical for optimizing computational performance across various domains. In server workloads, hardware data prefetching significantly enhances performance in applications managing extensive datasets by effectively reducing latency and improving resource utilization [17]. In cloud computing environments, data prefetching techniques align with the intrinsic characteristics of workloads, ensuring efficient processing and adherence to Service Level Agreements (SLAs).

Speculative execution is vital in high-performance computing, predicting and executing future instruction paths to enhance throughput, particularly in deadline-sensitive cloud processing systems where SLA maintenance is crucial [18]. It is also employed in distributed computation to mitigate stragglers' impact through redundant task execution, optimizing overall system performance [19].

In natural language processing, speculative execution techniques such as blockwise parallel decoding and speculative decoding have shown promise in accelerating the decoding speed of large language models (LLMs), improving inference efficiency and reducing latency in complex computational tasks [4]. The widespread implementation of these techniques across sectors, including cloud computing and data-intensive scientific workflows, underscores their essential contribution to enhancing computational efficiency and addressing memory latency challenges, significantly improving high-demand applications such as media streaming, web search, and large-scale data processing services [20, 21, 17]. These techniques effectively tackle contemporary performance optimization challenges, underscoring their indispensability in a rapidly evolving technological landscape.

1.4 Structure of the Survey

This survey is meticulously structured to provide a comprehensive exploration of data prefetching and speculative execution techniques and their pivotal role in hardware optimization. The paper begins with an **Introduction** section, highlighting the significance of these techniques in minimizing latency and enhancing processor throughput, followed by an examination of the challenges faced by modern computing systems and the relevance of these techniques across diverse applications. The **Background and Definitions** section offers a detailed explanation of core concepts, setting the stage for in-depth discussions in subsequent sections.

The survey delves into , examining a range of strategies designed to enhance memory access efficiency. These strategies include software-only methods, learning-based approaches such as reinforcement learning, dynamic and adaptive techniques that adjust to varying workloads, and hybrid models that combine multiple prefetching mechanisms. Each approach aims to mitigate the latency associated with memory accesses by predicting future data requirements and preloading data into cache, thereby improving overall system performance in diverse application scenarios [17, 22, 20, 23, 24]. This is followed by an examination of the **Runahead Mechanism**, detailing its implementations, challenges, and performance benefits. The role of **Instruction Dataflow Classification** in enhancing parallel execution through data dependency management is then analyzed.

Subsequently, the survey delves into **Hardware Optimization Techniques**, discussing the integration of static, dynamic, and speculative methods, along with compiler and co-design approaches, and innovative architectures. The **Cache Performance** section evaluates cache performance metrics and optimization techniques, providing case studies and experimental evaluations.

The exploration of **Speculative Execution** addresses its potential for performance improvement and associated security concerns, referencing works such as [25] and [26], which discuss detection and mitigation of vulnerabilities. Finally, the **Conclusion** synthesizes the survey's findings, emphasizing the importance of data prefetching and speculative execution in hardware optimization and suggesting future research directions.The following sections are organized as shown in Figure 1.

2 Background and Definitions

2.1 Background and Definitions

Data prefetching is a critical technique designed to minimize memory latency by preemptively loading data into the cache before processor requests, thereby improving cache efficiency and reducing access time for frequently accessed data. This optimization is crucial for overall system performance, particularly in cloud environments where memory access patterns are analyzed to refine prefetching strategies [10]. Techniques such as ReVeLA leverage program semantics from vectorized codes to predict memory accesses accurately and generate effective prefetch requests [27]. Dynamic frameworks like RPG 2 profile running programs to identify cache misses, injecting optimized prefetch instructions and adjusting parameters based on real-time metrics [28]. Cache Level Prediction further minimizes load latency by predicting the memory hierarchy level a load will access, enabling earlier memory load initiation [29].

The runahead mechanism enhances instruction-level parallelism (ILP) by allowing processors to execute instructions during cache misses, thus improving throughput. It leverages cache miss periods for pre-execution, significantly boosting processor efficiency in handling subsequent instructions. This approach is essential in maximizing ILP, as demonstrated in advanced cache management strategies like the Least Hit Density (LHD) eviction policy, which optimizes cache hit rates through modeling object behavior, and query optimizers that exploit architecture-specific features for improved execution efficiency [3, 30, 31].

Instruction dataflow classification plays a pivotal role by categorizing instructions based on data dependencies, facilitating parallel execution and enhancing hardware optimization. This classification identifies independent instructions for concurrent execution, maximizing ILP and processor efficiency. The classification of data prefetching techniques based on memory access patterns—strided, temporal, and spatial correlations—illustrates mechanisms addressing specific performance challenges [17].

Hardware optimization entails various strategies aimed at improving the efficiency and performance of hardware components. Techniques like Compute Cache enable in-place computation by transforming cache elements into computational units, allowing operations directly on cached data without data transfer [32]. The SNCPU architecture exemplifies hardware optimization, functioning as both a RISC-V CPU and a systolic CNN accelerator, streamlining data flow for deep learning tasks [33]. The Split Latency Adaptive Pipeline (SLAP) architecture enhances VLIW architecture by enabling independent execution of scalar and vector pipelines, boosting performance in wireless baseband applications [34].

Cache performance is evaluated by its effectiveness in reducing access time to frequently used data. Techniques such as Cagra, which employs CSR Segmenting to enhance cache utilization in graph analytics, highlight the importance of optimizing cache performance [35]. Extending TTL cache models to incorporate random object fetch delays provides a more realistic analysis of cache performance metrics, accommodating variations in network delays [36]. Leeway, a dead block predictor for LLC, uses Live Distance, derived from stack distance, to determine a cache block's useful lifetime, optimizing cache management [37].

Speculative execution predicts and executes future instruction paths in advance, potentially boosting performance by increasing throughput. However, it requires mechanisms to manage incorrect predictions to maintain system integrity. Speculative execution can expose sensitive data, leading to vulnerabilities; mechanisms like Okapi enforce secure speculative execution [38]. SpecDec utilizes speculative execution to accelerate seq2seq generation by drafting multiple tokens in parallel and verifying them efficiently [39]. Frameworks like Chronos integrate various speculative execution strategies to optimize the completion probability of deadline-critical MapReduce jobs [18].

These foundational concepts are essential for enhancing hardware performance by effectively addressing high memory latency through advanced techniques like data prefetching, which anticipates future memory accesses to minimize wait times. They are also crucial for optimizing resource utilization in modern computing systems, particularly in server environments where workloads demand rapid data processing and low-latency responses [40, 41, 23, 17].

In recent years, the exploration of data prefetching techniques has gained significant attention in the field of computer architecture. These techniques are crucial for enhancing cache performance and minimizing latency, which are vital for improving overall system efficiency. To provide a clearer understanding of the various approaches to data prefetching, Figure 2 illustrates a hierarchical classification of these techniques. This figure categorizes the techniques into four main groups: software-only, learning-based, dynamic and adaptive, and hybrid and holistic approaches. Each category encompasses specific frameworks, methods, and strategies aimed at optimizing cache performance and reducing latency through various innovative techniques. This structured overview not only highlights the diversity of approaches but also underscores the ongoing advancements in the field.

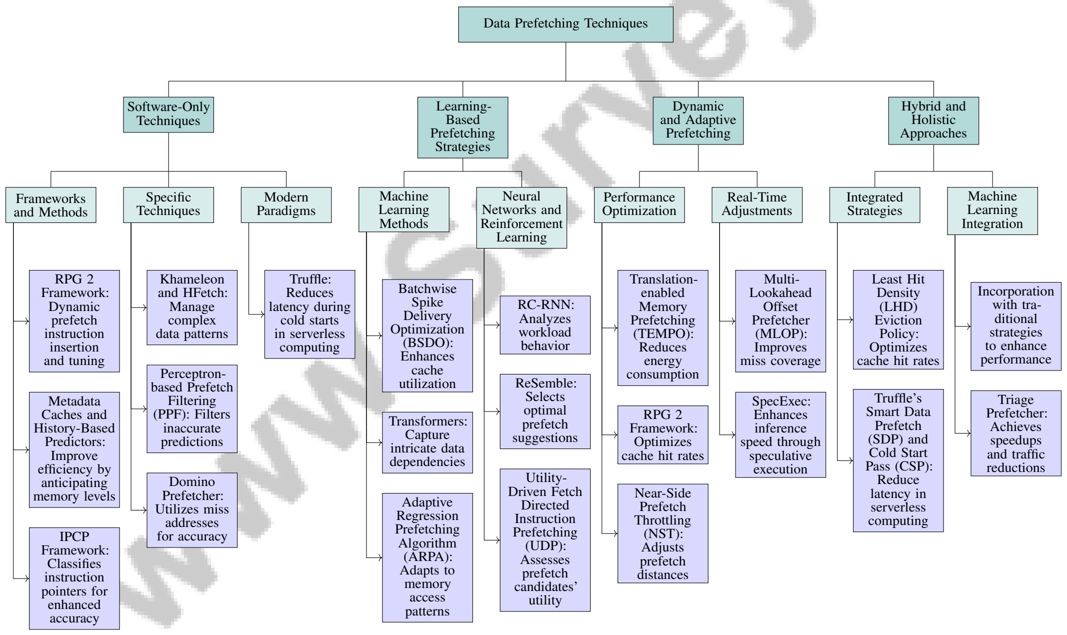

Data Prefetching Techniques
Software-Only Techniques
Learning-Based Prefetching Strategies
Dynamic and Adaptive Prefetching
Hybrid and Holistic Approaches
Frameworks and Methods
Specific Techniques
Modern Paradigms
Machine Learning Methods
Neural Networks and Reinforcement Learning
Performance Optimization
Real-Time Adjustments
Integrated Strategies
Machine Learning Integration
RPG 2 Framework: Dynamic prefetch instruction insertion and tuning
Metadata Caches and History-Based Predictors: Improve efficiency by anticipating memory levels
IPCP Framework: Classifies instruction pointers for enhanced accuracy
Khameleon and HFetch: Manage complex data patterns
Perceptron-based Prefetch Filtering (PPF): Filters inaccurate predictions
Domino Prefetcher: Utilizes miss addresses for accuracy
Truffle: Reduces latency during cold starts in serverless computing
Batchwise Spike Delivery Optimization (BSDO): Enhances cache utilization
Transformers: Capture intricate data dependencies
Adaptive Regression Prefetching Algorithm (ARPA): Adapts to memory access patterns
RC-RNN: Analyzes workload behavior
ReSemble: Selects optimal prefetch suggestions
Utility-Driven Fetch Directed Instruction Prefetching (UDP): Assesses prefetch candidates' utility
Translation-enabled Memory Prefetching (TEMPO): Reduces energy consumption
RPG 2 Framework: Optimizes cache hit rates
Near-Side Prefetch Throttling (NST): Adjusts prefetch distances
Multi-Lookahead Offset Prefetcher (MLOP): Improves miss coverage
SpecExec: Enhances inference speed through speculative execution
Least Hit Density (LHD) Eviction Policy: Optimizes cache hit rates
Truffle's Smart Data Prefetch (SDP) and Cold Start Pass (CSP): Reduce latency in serverless computing
Incorporation with traditional strategies to enhance performance
Triage Prefetcher: Achieves speedups and traffic reductions

Figure 2: This figure illustrates a hierarchical classification of data prefetching techniques, categorized into software-only, learning-based, dynamic and adaptive, and hybrid and holistic approaches. Each category encompasses specific frameworks, methods, and strategies aimed at optimizing cache performance and reducing latency through various innovative techniques.

3 Data Prefetching Techniques

3.1 Software-Only Techniques

Software-only data prefetching techniques enhance cache performance by strategically predicting and preloading data, reducing latency and optimizing memory use. The RPG 2 framework exemplifies this by dynamically inserting and tuning prefetch instructions during execution, providing adaptability

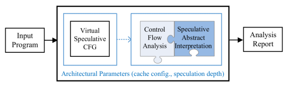
Method Name	Technique Type	Optimization Focus	Adaptability
RPG2[28]	Software-only	Improving Cache Performance	Dynamically Adjust
LP[29]	Software-only	Reducing Latency	Dynamically Insert
IPCP[42]	Software-only	Improving Cache Performance	Adapt TO Patterns
PPF[23]	Filtering Mechanism	Improving Cache Performance	Varying Workloads
Domino[9]	Temporal Prefetching	Improving Cache Performance	Adjust TO Workloads
T[2]	Modular Architecture	Reducing Latency	Varying Cold Start
AI-SE[43]	Static Analysis	Reduce Latency	Varying Workloads
N/A[38]	Hardware/software	Mitigating Tes	Dynamically Managing Permissions
WMM[44]	Load-value Speculation	Reduces Complexity	Simplifies Memory Model

Table 1: Overview of various data prefetching techniques, categorized by method name, technique type, optimization focus, and adaptability. This table highlights the diversity in software-only strategies and their dynamic adaptability to enhance cache performance and reduce latency.

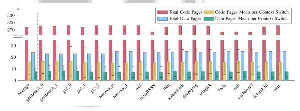
to varying workloads [28]. Techniques such as metadata caches and history-based predictors further improve efficiency by anticipating memory levels, reducing load latency [29]. The IPCP framework refines software-based prefetching by classifying instruction pointers into distinct categories, enhancing accuracy [42].

Khameleon and HFetch illustrate the role of software techniques in managing complex data patterns, optimizing response quality and latency through progressive encoding and hierarchical read operations, respectively [45, 21]. The Perceptron-based Prefetch Filtering (PPF) technique enhances prefetching by filtering inaccurate predictions, balancing coverage and accuracy [23]. Similarly, the Domino prefetcher improves accuracy by utilizing miss addresses, showcasing software techniques' potential in refining data access strategies [9].

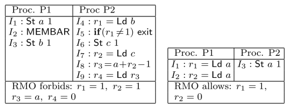
In serverless computing, Truffle reduces latency by enabling data transfer during cold starts, highlighting software-based prefetching's relevance in modern paradigms [2]. These techniques collectively demonstrate the significant impact of algorithmic innovations on cache performance, adapting to data access patterns. Table 1 provides a comprehensive comparison of software-only data prefetching techniques, illustrating their distinct approaches to optimizing cache performance and adaptability in varying computational environments.


Input Program
Virtual Speculative CFG
Control Flow Analysis
Speculative Abstract Interpretation
Analysis Report
Architectural Parameters (cache config., speculation depth)

(a) Virtual Speculative CFG Analysis[43]



(b) Comparison of Code Pages and Data Pages Across Different Programs and Context Switches[38]



Proc. P1	Proc P2
I_1 : St a 1 I_2 : MEMBAR I_3 : St b 1	$I_4 : r_1 =$ Ld b I_5 : **if**$(r_1 \neq 1)$ exit I_6 : St c 1 $I_7 : r_2 =$ Ld c $I_8 : r_3 = a + r_2 - 1$ $I_9 : r_4 =$ Ld r_3
RMO forbids: $r_1 = 1,\ r_2 = 1$ $r_3 = a,\ r_4 = 0$	

Proc. P1	Proc. P2
$I_1 : r_1 =$ Ld a $I_2 : r_2 =$ Ld a	I_3 : St a 1
RMO allows: $r_1 = 1,$ $r_2 = 0$	

(c) Table comparing two processes, P1 and P2, with their respective initial states and RMO outcomes.[44]

Figure 3: Examples of Software-Only Techniques

Figure 3 illustrates software-only approaches in data prefetching, emphasizing speculative analysis and context-aware evaluations that enhance computational efficiency [43, 38, 44].

3.2 Learning-Based Prefetching Strategies

Method Name	Machine Learning Models	Optimization Techniques	Prefetching Strategy Types
BSDO[46]	Group Prefetching	Batch Processing	Adaptive Regression
DLPP[47]	Transformer Models	Dynamic Analysis	Adaptive Regression
ARPA[48]	Adaptive Regression Prefetching	Optimized Linear Regression	Adaptive Regression Prefetching
RNN[11]	Recurrent Neural Networks	Dynamic Caching Architecture	Adaptive Regression
RL-EPF[22]	Multilayer Perceptron	Dynamic Selection Mechanism	Adaptive Ensemble Framework
UDP[13]	-	Dynamic Analysis	Adaptive Regression
PPF[23]	Perceptron-based Model	Spike Delivery Optimization	Perceptron-based Filtering

Table 2: Overview of learning-based prefetching strategies detailing the machine learning models, optimization techniques, and prefetching strategy types employed by various methods. This table highlights the diverse approaches and algorithms used to enhance cache performance and memory access through predictive data access pattern analysis.

Learning-based prefetching strategies employ machine learning to predict data access patterns, optimizing memory access and cache performance. The Batchwise Spike Delivery Optimization (BSDO) method restructures spike delivery to enhance cache utilization [46]. Deep learning models, such as Transformers, advance page prefetching by capturing intricate data dependencies, improving prediction accuracy [47].

The Adaptive Regression Prefetching Algorithm (ARPA) uses machine learning for dynamic analysis, adapting to memory access patterns [48]. Recurrent Neural Networks (RNNs), as in the RC-RNN method, optimize caching decisions by analyzing workload behavior [11]. Reinforcement learning, exemplified by ReSemble, selects optimal prefetch suggestions from multiple prefetchers, maximizing rewards from hits and misses [22].

Utility-Driven Fetch Directed Instruction Prefetching (UDP) optimizes instruction prefetching by assessing prefetch candidates' utility [13]. Perceptron-based models, like the Perceptron-based Prefetch Filtering (PPF) technique, predict and filter inaccurate prefetches, enhancing prefetcher tuning [23]. These methods underscore machine learning's role in refining prefetching strategies, balancing coverage and accuracy.

As illustrated in Figure 4, the hierarchical structure of learning-based prefetching strategies is categorized into machine learning models, neural networks, and prefetch optimization techniques. This figure highlights key methods and their respective contributions to optimizing cache performance and memory access, emphasizing the importance of these strategies in advancing data prefetching. Table 2 provides a comprehensive summary of learning-based prefetching strategies, elucidating the different machine learning models, optimization techniques, and prefetching strategy types utilized to improve cache and memory performance.

Learning-based strategies represent a crucial advancement in data prefetching, leveraging algorithms to predict and adapt to complex data patterns. Methods such as perceptron learning and reinforcement learning enhance cache performance, improving instructions per cycle (IPC) and execution performance [23, 22, 24].

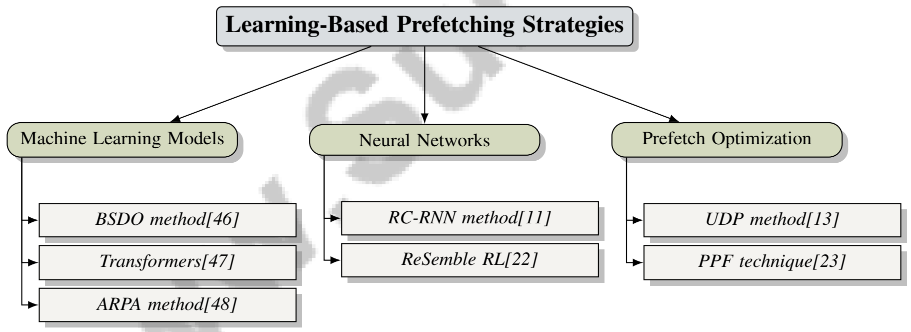


Figure 4: This figure illustrates the hierarchical structure of learning-based prefetching strategies, categorized into machine learning models, neural networks, and prefetch optimization techniques, highlighting key methods and their respective contributions to optimizing cache performance and memory access.

3.3 Dynamic and Adaptive Prefetching

Dynamic and adaptive prefetching techniques optimize memory access by adjusting strategies based on real-time behavior, which is crucial for modern memory-intensive workloads. Translation-enabled memory prefetching (TEMPO) enhances performance and reduces energy consumption by predicting and prefetching data based on page table accesses [24, 41]. Advanced perceptron learning methods decrease unnecessary requests, mitigating cache pollution and improving performance [24].

The RPG 2 framework dynamically tunes prefetch strategies, optimizing cache hit rates across diverse workloads [28]. Near-Side Prefetch Throttling (NST) adjusts prefetch distances based on detection, optimizing bandwidth and reducing unnecessary prefetches [49]. The Multi-Lookahead Offset Prefetcher (MLOP) assigns scores to prefetch offsets, improving miss coverage and accuracy [3]. SpecExec, a speculative execution method, enhances inference speed by enabling parallel processing, showcasing dynamic techniques' role in performance improvement [50].

These techniques emphasize real-time adjustments' critical role in prefetching strategies, enhancing cache performance by mitigating bottlenecks and optimizing efficiency [49, 41, 17]. By adapting to runtime behavior, these methods ensure responsive prefetching, enhancing system efficiency and reducing latency.

Figure 5 illustrates the key dynamic and adaptive prefetching techniques discussed in the text, highlighting the main categories and specific methods that optimize memory access by adjusting strategies based on real-time behavior. This visual representation serves to reinforce the narrative by providing a clear overview of the relationships and methodologies that contribute to effective memory management.

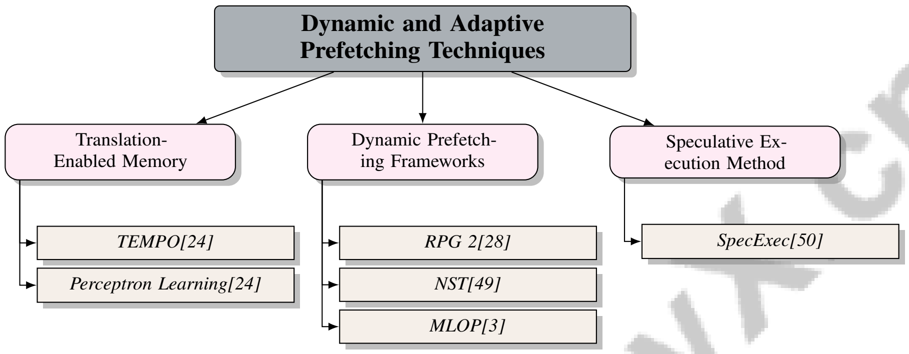

Dynamic and Adaptive Prefetching Techniques
Translation-Enabled Memory
Dynamic Prefetching Frameworks
Speculative Execution Method
TEMPO[24]
Perceptron Learning[24]
RPG 2[28]
NST[49]
MLOP[3]
SpecExec[50]

Figure 5: This figure illustrates the key dynamic and adaptive prefetching techniques discussed in the text, highlighting the main categories and specific methods that optimize memory access by adjusting strategies based on real-time behavior.

3.4 Hybrid and Holistic Approaches

Hybrid and holistic approaches in data prefetching and speculative execution integrate multiple strategies to optimize system performance, enhancing cache utilization and reducing latency. The least hit density (LHD) eviction policy dynamically models object behavior to optimize cache hit rates, while TEMPO leverages the virtual memory subsystem to reduce DRAM access times [41, 30].

Truffle's Smart Data Prefetch (SDP) and Cold Start Pass (CSP) mechanisms exemplify hybrid strategies by integrating data retrieval with function initialization, reducing latency in serverless computing [2]. In speculative execution, benchmarks for Ethereum smart contracts explore strategies maximizing throughput while ensuring data consistency, illustrating holistic approaches' effectiveness in managing dependency issues [8].

Hybrid approaches increasingly incorporate machine learning with traditional strategies, enhancing cache performance by reducing unnecessary requests and adapting to workload characteristics [51, 23, 22]. These methods improve cache hit ratios and ensure real-time adaptability, showcasing hybrid strategies' effectiveness in optimizing performance.

Hybrid and holistic approaches demonstrate significant promise in enhancing cache performance and system efficiency. Techniques like the Triage prefetcher achieve notable speedups and traffic reductions, while MLOP demonstrates substantial performance improvements [52, 41, 3, 17]. By integrating multiple strategies, these approaches effectively address modern computing challenges, emphasizing adaptability and comprehensive strategy in managing data access patterns.

4 Runahead Mechanism

The Runahead Mechanism is a pivotal strategy in modern computing architectures, designed to overcome latency issues associated with cache misses, thus enhancing instruction-level parallelism (ILP) and processor throughput. By addressing frontend stalls due to instruction cache misses through advanced prefetching techniques, such as fetch-directed prefetching, it achieves an average IPC improvement of 3.6

4.1 Overview of Runahead Execution

Runahead execution is essential in modern processors to mitigate performance penalties from cache misses by maintaining ILP and enhancing throughput. It allows CPUs to execute instructions even when data is unavailable in the cache, crucial for modern applications that often exceed cache sizes, causing frontend stalls. By anticipating and prefetching instructions and data, runahead execution reduces the impact of long-latency memory accesses, optimizing system performance [49, 3, 29, 13]. Through speculative execution during cache misses, it converts idle cycles into productive computation, reducing memory access wait times.

Speculative execution frameworks, like those in large language models (LLMs), illustrate runahead execution's utility by enhancing decoding speed and throughput [4]. By speculatively executing instructions, processors resolve dependencies and prepare data paths for future execution, ensuring a steady instruction flow.

The IMP-ro-spec method exemplifies speculative execution constructs in modeling global state changes, supporting practical runahead execution implementations [7]. This modeling is crucial for optimizing speculative execution's effects on processor performance, enabling efficient handling of complex tasks.

Runahead execution enhances ILP and throughput by utilizing cache miss intervals for pre-execution of instructions, reducing idle periods and increasing system performance. Advanced prefetching techniques aim to balance coverage and accuracy, minimizing resource waste and improving execution speed in memory-intensive applications [41, 13, 23, 24, 3].

4.2 Techniques and Implementations

Runahead execution techniques enhance processor throughput by enabling speculative execution during cache misses, maintaining ILP and improving data access efficiency. These techniques utilize strategies like the Shot Noise Model (SNM) for caching performance analysis, on-chip temporal prefetchers to reduce off-chip metadata latency, and Scheduling-Aware Data Prefetching (SADP) for efficient memory management in cloud services [52, 53, 20].

Speculative execution methods, such as SCA, optimize job utility and minimize resource consumption by scheduling extra task copies in lightly loaded clusters, enhancing throughput in distributed environments [54]. An empirical study in Ethereum smart contracts reveals a method using concurrent phases for non-conflicting transactions, leveraging speculative execution to manage data dependencies and enhance throughput [8].

ReSemble optimizes prefetching decisions by learning from prefetcher-cache interactions in real-time [22]. The MRT technique ensures speculative execution reliability by comparing contract and hardware traces [55].

In GPU-CPU interactions, a novel synchronization mechanism allows CPU control over GPU scheduling, maintaining optimal runahead distance and enhancing temporal locality [56]. The SLNO method integrates speculative techniques with loop transformations, enabling processors to handle data-dependent flows effectively [57].

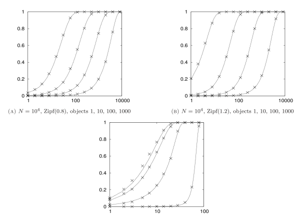
The Triage method achieves significant performance improvements, with a 23.5% speedup over the Best Offset Prefetcher [52]. The Domino prefetcher improves accuracy using recent miss addresses, highlighting innovative prefetching strategies' role in optimizing data access [9].

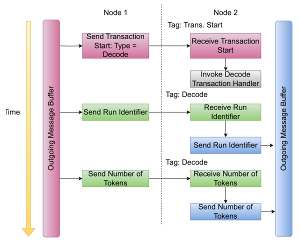
InvarSpec identifies speculation invariant instructions, optimizing speculative execution and improving efficiency [58]. This approach reduces speculative execution overhead, ensuring effective resource utilization.

These techniques highlight runahead execution's versatility and effectiveness in modern systems. By employing speculative execution and adaptive strategies, these methods significantly increase throughput, enhance data access efficiency, and elevate system performance. Speculative Decoding achieves up to a 5× speedup in autoregressive tasks. Optimization strategies in parallel frameworks, like Smart Cloning and Enhanced Speculative Execution, reduce job flow times by nearly 60

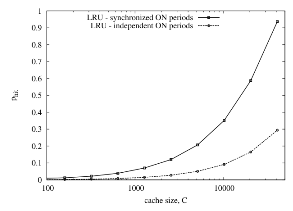
As illustrated in Figure 6, the Runahead Mechanism is crucial for enhancing processing efficiency and system performance. These examples demonstrate various techniques and implementations that align

AI-generated, for reference only.


(a) Comparison of Zipfian distributions with different parameters and object sizes[59]



(b) A Flowchart of a Transaction Processing System[60]



(c) The graph compares the probability of hit (Phit) for two different cache management strategies: LRU (Least Recently Used) with synchronized ON periods and LRU with independent ON periods.[61]

Figure 6: Examples of Techniques and Implementations

with the runahead concept in optimizing system operations. The comparison of Zipfian distributions illustrates how data distribution influences efficiency, akin to runahead execution prefetching to mitigate latency. The transaction processing system flowchart highlights the coordination necessary in complex operations, reflecting runahead mechanisms' predictive nature. The cache management strategies graph emphasizes strategic data handling, showcasing runahead's ability to improve cache utilization through preemptive instruction processing. These examples underscore the diverse applications and implementations of techniques that, like the runahead mechanism, streamline and accelerate computational processes [59, 60, 61].

4.3 Challenges and Vulnerabilities

Despite its benefits in enhancing ILP and throughput, runahead execution introduces challenges and vulnerabilities, particularly in speculative execution. Notably, security vulnerabilities arise from microarchitectural side channels, where speculative execution paths may expose sensitive data. This issue is exacerbated by inadequate countermeasures for Spectre attacks, which do not fully address vulnerabilities from Return Stack Buffer (RSB) mispredictions [62].

The divergence of real-world operating systems from controlled research environments complicates effective defense implementation. This divergence, especially in user and kernel space interactions, limits existing methods' applicability, as shown in mitigating Spectre-like vulnerabilities [5].

Moreover, the lack of a comprehensive evaluation framework hinders effective comparison and assessment of speculative execution approaches, complicating robust solution development [4]. This gap highlights the need for standardized methodologies to evaluate speculative execution techniques comprehensively.

Speculative execution can create vulnerabilities known as cache timing leaks, where speculative execution alters the cache state, allowing attackers to infer sensitive information through timing variances. By executing operations not typically occurring in a functioning program, adversaries can access confidential data via side-channel attacks, undermining security mechanisms like process separation and just-in-time compilation. This threat is significant, as speculative execution is present in microprocessors from major manufacturers like Intel, AMD, and ARM, used in billions of devices [63, 64, 65, 66, 16]. These leaks necessitate novel detection and mitigation strategies that account for speculative execution's transient nature.

Furthermore, speculative execution mechanisms' performance overhead, such as filter caches, can degrade performance under certain workloads, highlighting the trade-offs between security and performance [67]. Existing detection methods often face limitations, necessitating innovative approaches to identify and mitigate speculative execution vulnerabilities [26].

Addressing runahead execution's challenges and vulnerabilities requires ongoing research and holistic solutions that balance security, performance, and adaptability in speculative execution environments. This includes addressing security implications revealed by speculative execution studies, such as side-channel attacks like Spectre and Meltdown, which exploit speculative computations' transient nature to leak sensitive information. Advancements in frameworks like SPEECHMINER underscore the need for systematic approaches to quantify and mitigate these vulnerabilities across microprocessor architectures, ensuring hardware designs and software practices evolve to safeguard against speculative execution threats [63, 7, 64, 68].

4.4 Performance Enhancements

Runahead execution significantly enhances system performance by leveraging speculative execution to optimize ILP and throughput, converting idle cycles during cache misses into productive computation time. This improves overall system efficiency. Speculative execution in visual analytics enhances user experience by facilitating intuitive model exploration and optimization [69].

Speculative execution techniques, like SpecExec, have achieved remarkable speed improvements, up to 18.7x over traditional methods [50], demonstrating their potential to enhance processing speed and efficiency, especially in rapid data analysis and decision-making scenarios.

The NDA framework mitigates all known speculative execution attacks while preserving much of the performance advantage of out-of-order execution, providing significant performance enhancements [70]. This demonstrates speculative execution's capability to deliver performance benefits without compromising security.

In graph analytics, the Cagra framework achieves up to a 5x speedup for various applications compared to the best published results, highlighting significant improvements in cache efficiency and overall performance [35]. Such advancements underscore speculative execution's transformative impact on cache performance and system throughput.

SafeSpec design effectively prevents all known Meltdown and Spectre variants, demonstrating negligible performance impact and even modest improvements in some scenarios [71]. This highlights the dual benefit of enhancing security while maintaining or improving performance.

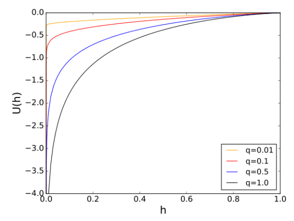
In distributed computing, speculative execution strategies yield significant speed-ups, although these benefits may diminish with increasing transaction volumes and conflict rates [8]. Nonetheless, achieving substantial performance gains in distributed systems illustrates speculative execution's transformative potential on modern architectures.

Innovative caching strategies that analyze dynamic content popularity have proven both accurate and computationally efficient, capturing content dynamics without compromising efficiency [61]. This method's efficiency is crucial for optimizing cache performance in environments with fluctuating access patterns.

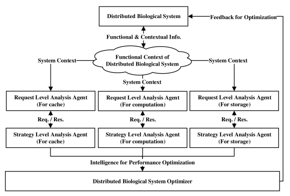
Runahead execution and speculative execution techniques significantly improve system performance by optimizing resource utilization, enhancing processing speed, and addressing security concerns. Advancements like Speculative Decoding achieve up to a 5× speedup in autoregressive tasks, and algorithms effectively mitigate the straggler problem in parallel processing [63, 64, 72, 54, 39]. These advancements underscore speculative execution's critical role in modern systems, highlighting its potential to drive future performance optimization innovations.

Query	Speedup	Query	Speedup
Q1	2.84	Q12	2.67
Q2	1.65	Q13	1.94
Q3	1.21	Q14	2.08
Q4	1.39	Q15	4.09
Q5	1.33	Q16	1.37
Q6	3.64	Q17	3.22
Q7	2.79	Q18	2.68
Q8	1.63	Q19	1.71
Q9	1.28	Q20	1.38
Q10	1.62	Q21	1.55
Q11	1.98	Q22	1.39

(a) The table shows the speedup of queries in a system.[31]



(b) U(h)[73]


Distributed Biological System
Feedback for Optimization
Functional & Contextual Info.
Functional Context of Distributed Biological System
System Context
System Context
System Context
Request Level Analysis Agent (For cache)
Request Level Analysis Agent (For computation)
Request Level Analysis Agent (For storage)
Req. / Res.
Req. / Res.
Req. / Res.
Strategy Level Analysis Agent (For cache)
Strategy Level Analysis Agent (For computation)
Strategy Level Analysis Agent (For storage)
Intelligence for Performance Optimization
Distributed Biological System Optimizer

(c) Distributed Biological System Optimizer[74]

Figure 7: Examples of Performance Enhancements

As shown in Figure 7, the runahead mechanism is pivotal in enhancing computational system performance. This is illustrated through three examples, each highlighting different aspects of performance optimization. A table showcases query speedup, with values from 1.21 to 4.09, underscoring the runahead mechanism's capability to expedite processing. The graph "U(h)" presents performance variations across different q values, offering insights into parameter adjustments' influence on performance. The Distributed Biological System Optimizer flowchart demonstrates optimizing distributed systems by integrating functional and contextual information, emphasizing cache, computation, and storage request analysis. Collectively, these examples provide a robust understanding of how runahead techniques optimize system performance across diverse applications [31, 73, 74].

5 Instruction Dataflow Classification

Understanding instruction dataflow classification is essential for analyzing interactions and dependencies among instructions, which is crucial for enhancing execution efficiency and addressing data dependency challenges. This section delves into the foundational aspects of instruction dataflow, setting the stage for examining its application within the KV-Runahead method. The subsequent subsection highlights its practical implications in optimizing large language model inference tasks.

5.1 Instruction Dataflow in KV-Runahead Method

The KV-Runahead method leverages instruction dataflow classification to partition user context into segments, optimizing large language model (LLM) inference [75]. This approach mitigates Instruction-Level Parallelism (ILP) limitations caused by control and true data dependencies. By employing effective prediction mechanisms, KV-Runahead enhances parallel execution and throughput [72].

Incorporating instruction dataflow classification within the KV-Runahead framework enables the identification and categorization of instructions based on data dependencies, allowing for concurrent execution of independent instructions. This maximizes ILP and enhances processor efficiency, aligning with parallelized sequential composition principles and improving programming languages' expressiveness in managing weak memory behaviors [76].

KV-Runahead optimizes LLM inference by utilizing instruction dataflow classification to streamline execution flow. It manages data dependencies and maximizes throughput by parallelizing the prompt phase of inference. Through orchestrating multiple processes to populate the key-value cache (KV-cache), KV-Runahead reduces time-to-first-token (TTFT) and accelerates token generation during the extension phase. Context-level load balancing addresses uneven KV-cache generation, resulting in speedups exceeding 1.4× for Llama 7B and 1.6× for Falcon 7B compared to traditional methods [77, 60, 75, 50]. This highlights the critical role of dataflow classification in boosting hardware performance in high-parallelism environments.

5.2 Instruction-Level Parallelism (ILP) and Data Dependency

Instruction-level parallelism (ILP) is central to modern processor architectures, enabling simultaneous execution of multiple instructions to enhance computational throughput. However, ILP effectiveness is often constrained by data dependencies, which dictate instruction execution order to maintain data integrity. This challenge is heightened in processors using speculative execution, where out-of-order execution can introduce unexpected behaviors and security vulnerabilities, necessitating advanced static analysis for accurate modeling and optimization [7, 78, 76, 43]. Dataflow classification is crucial in managing these dependencies by identifying independent instructions for concurrent execution.

Out-of-order execution and instruction reordering complexities, as discussed in [76], complicate program execution consistency and correctness. Dataflow classification addresses these challenges by categorizing instructions based on data dependencies, facilitating parallel execution of non-dependent instructions and optimizing ILP.

Employing dataflow classification techniques allows processors to manage instruction dependencies effectively, enabling parallel execution of independent instructions while ensuring proper order for dependent instructions. This enhances ILP and mitigates the "memory wall" problem, as demonstrated by advanced hardware prefetching mechanisms like the Instruction Pointer Classifier-based spatial Prefetching (IPCP), which optimizes data access patterns in cache hierarchies [42, 72, 76].

Integrating dataflow classification into processor architectures is crucial for optimizing performance in environments with complex data dependencies and high demands for parallel execution.

6 Hardware Optimization Techniques

6.1 Integration of Static, Dynamic, and Speculative Techniques

Integrating static, dynamic, and speculative techniques in hardware optimization forms a comprehensive strategy for enhancing system performance. Static techniques analyze program behavior at compile time, identifying bottlenecks and applying optimizations effective regardless of runtime conditions [57]. Dynamic techniques adapt to runtime conditions, enhancing flexibility and responsiveness to varying workloads, as demonstrated in query optimization processes [31]. Speculative techniques predict and execute future instruction paths, increasing throughput and reducing latency, with frameworks like CoSpec optimizing intermittent computing without costly hardware changes [79]. SpecControl further refines speculative execution by implementing fine-grained restrictions based on true dependencies, achieving notable performance improvements [80].

Colvin et al. propose a framework that abstracts speculative execution complexities while addressing security implications, particularly regarding cache behaviors [7]. Unified attack detection models achieving over 99

The integration of these techniques is vital for optimizing hardware performance, ensuring efficient resource utilization, and maintaining security in evolving computing environments. This strategy emphasizes the importance of advanced data prefetching and architecture-specific query optimization in improving modern hardware systems [40, 17, 31].

6.2 Compiler and Co-Design Approaches

Compiler and co-design approaches play a crucial role in hardware optimization by bridging software and hardware to enhance performance and efficiency. These approaches utilize advanced compiler techniques for code analysis and optimizations, particularly in speculative execution, crucial for execution time estimation and side channel detection [57, 78, 81, 43]. Co-design strategies foster collaborative development, enabling effective integration of security mechanisms and performance optimizations, addressing challenges like data locality and synchronization in parallel processing environments.

The Declassiflow model enhances security and performance by statically identifying non-secret data, mitigating speculative execution attacks [82]. SpecControl demonstrates the adaptability of co-design approaches by implementing secure speculation policies without hardware modifications, highlighting the potential of co-design strategies to enhance security and performance [80].

Compiler and co-design approaches enable tailored optimization strategies for various hardware architectures, addressing challenges such as data locality and synchronization. This targeted optimization maximizes performance gains through static, dynamic, and speculative loop nest transformations, adapting to the evolving landscape of computer architecture [57, 42, 31]. Innovative hardware prefetching techniques, like IPCP, achieve substantial performance improvements in memory-intensive applications by addressing the "memory wall" problem. The development of hardware-software contracts for secure speculation enhances security while maintaining efficiency, balancing speculative execution risks. These advancements underscore the essential nature of compiler and co-design strategies in optimizing modern computing technologies [42, 81, 17, 31].

6.3 Innovative Architectures and Frameworks

Innovative architectures and frameworks are pivotal in advancing hardware optimization by integrating cutting-edge techniques that enhance performance and efficiency. Virtual control flow models speculative execution, developing optimizations for managing merges and loops while bounding speculative execution depth [43]. The DSMT architecture employs dynamic simultaneous multithreading to optimize resource allocation and enhance throughput [1].

Empirical studies of speculative concurrency emphasize using historical transaction data to inform speculative execution strategies, offering realistic performance assessments [8]. Frameworks like

AI-generated, for reference only.

SafeBet focus on securing speculative execution while maintaining simplicity and speed, with future research aimed at enhancing its efficiency [83]. The integration of SpecEx into Visual Analytics frameworks represents another frontier, emphasizing speculative execution-guided visual analytics in optimizing computational processes [69].

Extending methods like cache simulation using polyhedral models to complex cache designs and integrating them with modern speculative execution models promises enhanced accuracy and efficiency [84]. Future research on the Parallelized Sequential Composition (PSC) method will extend its semantics to encompass features of modern processors, such as cache systems and memory models, refining its applicability in optimizing hardware performance [76].

Innovative architectures and frameworks are essential for optimizing hardware performance by employing advanced techniques such as data prefetching to mitigate latency. These approaches enhance performance, efficiency, and security, particularly in high-demand environments like data centers, where workloads require low latency and high-quality service. Developments like the CLIP load criticality predictor demonstrate significant improvements in prefetching accuracy, essential for maximizing memory system effectiveness in bandwidth-constrained configurations [40, 17]. These innovations underscore the critical role of forward-thinking design and integration strategies in advancing contemporary hardware architectures.

7 Cache Performance

7.1 Cache Performance Metrics

Benchmark	Size	Domain	Task Format	Metric
DPI[85]	1,000,000	High Performance Computing	Data Prefetching	Accuracy, Timeliness
Spectre-Bench[64]	1,000,000	Computer Security	Security Evaluation	Attack Success Rate, Mitigation Overhead
MLCR[51]	300,000	Cache Networks	Caching And Routing	Hit Rate, Delay
SPECULOSE[63]	5,000	Computer Security	Security Analysis	Execution Time, Cache Hit Rate

Table 3: Table illustrating representative benchmarks used for evaluating cache performance metrics, detailing their size, domain, task format, and metrics used for assessment. This comprehensive overview aids in understanding the diverse applications and evaluation criteria across different computational domains.

Cache performance metrics, such as hit rate and access times, are vital for assessing caching strategies in modern systems. These metrics reveal how well systems handle dynamic content popularity and request patterns while minimizing retrieval costs and optimizing resource use [30, 59, 61, 73]. The cache miss rate, indicating the frequency of data retrieval from slower memory, is crucial for evaluating caching strategies' effectiveness in reducing latency and enhancing throughput.

Table 3 provides a detailed overview of representative benchmarks utilized in the analysis of cache performance metrics, highlighting their relevance across various computational domains and tasks. Simulation techniques using polyhedral models provide a robust framework for analyzing cache performance across configurations and workloads [84]. These simulations allow for comprehensive assessments of speedups and accuracy by comparing cache miss counts across benchmarks. By examining these metrics, researchers can refine caching strategies and design more efficient cache architectures.

Metrics evaluating data access time and overall system speedup through caching illustrate the balance between cache size, access speed, and efficiency. They show how cache size variations affect hit rates under different workloads. Innovative eviction policies like least hit density (LHD), which optimizes hit rates by predicting object behavior, are particularly noteworthy [73, 59, 30].

Analyzing cache performance metrics is essential for optimizing caching strategies, identifying areas for improvement, and developing solutions tailored to dynamic content demands. Advanced models and algorithms enhance understanding of the interplay between cache hit rates, request timing, and content dynamics, ultimately improving performance in real-world caching systems [86, 53, 73, 59, 61].

7.2 Techniques for Optimizing Cache Performance

Enhancing cache performance is crucial for improving system efficiency by reducing latency and increasing data access speed. Strategies focus on boosting cache hit rates and minimizing memory access times. The q_i-LRU policy, for instance, effectively reduces HDD load and improves cache hit ratios [73]. The AWRP technique surpasses traditional methods like LRU, FIFO, and CAR, demonstrating superior optimization capabilities [87].

Learning-based approaches are critical in dynamic cache management. The Seer-MCache, evaluated on metrics like cache hit rates and query response times, showcases machine learning's role in optimizing cache performance [12]. The P-OPT strategy further exemplifies this, achieving a 33% improvement over LRU and 22% over DRRIP, highlighting machine learning's impact [88].

Innovative strategies like LHD dynamically predict object eviction based on hit density, enhancing performance without traditional heuristics [30]. The Leeway method improves efficiency by predicting dead blocks, demonstrating superior resource management [37].

Prefetching strategies significantly optimize performance by reducing latency and improving utilization, as shown by Bhattacharjee et al. [41]. The Chrome framework measures demand miss ratios and effective prefetch hit ratios, illustrating prefetching's role in optimization [89].

Integrating pipelined synchronization mechanisms, as demonstrated by Gerzhoy et al., reduces DRAM accesses by 30.4

7.3 Challenges in Cache Performance Evaluation

Evaluating cache performance is challenging, especially with modern systems characterized by diverse workloads. Variability in workloads can significantly impact cache metrics like hit rates and latency. Techniques like LHD, which predict evictions based on hit density, may not perform optimally in specialized workloads [30]. This variability necessitates adaptable caching strategies.

Accurate simulation and modeling of cache behavior under various configurations are also challenging. While polyhedral model simulations provide a framework for metric analysis, they must be calibrated to reflect real-world conditions [84]. Ensuring accuracy and relevance in simulation results is crucial for effective caching strategy design.

The integration of machine learning in cache management adds complexity to performance evaluation. Learning-based approaches show potential but require comprehensive training and validation to ensure reliability across scenarios. This is vital for optimizing hit rates and prefetching strategies, where rigorous evaluation is needed to ensure methods like LHD and perceptron-based filtering adapt well to varying workloads without incurring excessive resource costs [23, 30, 24]. The dynamic nature of these methods complicates consistency and reliability in evaluations.

Continuous research and innovation in methodologies are essential to address these challenges. Factors like content request locality, user freshness demands, and dynamic content popularity must be considered. Developing robust models like the Shot Noise Model (SNM) for realistic traffic representation, enhancing LRU performance approximations, and creating adaptive caching policies prioritizing content freshness are critical for optimizing strategies [86, 53, 59, 61]. Comprehensive evaluation frameworks are necessary for optimizing caching strategies to meet modern demands.

7.4 Case Studies and Experimental Evaluations

Case studies and experimental evaluations are crucial for understanding the practical impact of cache optimization techniques. The Locality-aware Hit Density (LHD) method, evaluated using commercial memcached and Microsoft Research traces, demonstrates superior performance compared to traditional policies [30].

Using real-world traces in evaluations underscores the importance of diverse datasets in cache performance studies. Analyzing traces from commercial and research environments provides insights into the adaptability of caching strategies to different workloads. This assessment is supported by models like the Shot Noise Model (SNM) and policies like LHD, which optimize performance by predicting object behavior in real time [53, 59, 61, 30].

Experimental evaluations often compare new techniques against benchmarks to quantify improvements in hit rates, latency reduction, and overall performance. These comparisons yield insights into the trade-offs and advantages of various strategies, especially regarding content popularity and real-time traffic conditions. By analyzing isolated and interconnected caches and employing innovative policies like LHD, this research informs effective cache management solutions for cloud applications [61, 30].

Case studies and experimental evaluations enhance understanding of cache optimization, particularly concerning content freshness and popularity. Research shows traditional policies often fail to meet freshness requirements, leading to sub-optimal performance. By analyzing strategies that account for user freshness and content popularity, more effective policies can be derived. Recent studies have introduced approaches prioritizing freshness and developed models capturing content popularity's impact on performance. These insights, validated through simulations, underscore the importance of empirical and theoretical investigations [86, 61]. Rigorous testing in real-world scenarios validates effectiveness and identifies areas for further refinement, contributing to more efficient computing systems.

8 Speculative Execution

8.1 Security Concerns

Speculative execution, while enhancing computational performance by anticipating future instruction paths, introduces significant security vulnerabilities due to microarchitectural side effects that can leak sensitive data. Transient execution attacks like Spectre, Meltdown, and Foreshadow exploit these vulnerabilities, underscoring the urgent need for robust security frameworks [6, 80]. These attacks expose fundamental design flaws in processors, revealing that the performance benefits of speculative execution may come at the cost of severe security breaches.

Accurate modeling of speculative execution is vital for identifying vulnerabilities and evaluating countermeasures. Techniques such as PROSPECT offer a provably secure speculation mechanism, mitigating transient execution risks by establishing clear security dependencies between authorization and access operations, thereby preventing race conditions exploited by attackers [6]. Mitigation strategies often face trade-offs between restricting speculative instruction execution and relying on hardware mechanisms lacking visibility into true branch dependencies, leading to performance overhead [80]. Solutions must balance security and performance by enabling fine-grained control over speculative execution paths.

Innovative methodologies like SpecuSym utilize speculative symbolic execution to analyze cache timing leaks, systematically exploring program state spaces and modeling speculative behaviors at conditional branches. This approach identifies potential leaks, detecting a range of cache timing leaks across various programs and configurations, outperforming traditional static analysis methods [63, 66, 90]. Enhancing kernel safety against speculative execution and side-channel attacks while maintaining acceptable performance overhead remains crucial for modern computing security.

Ongoing research is essential to address significant security vulnerabilities associated with speculative execution, particularly those exploited by Spectre and Meltdown. This research should focus on developing comprehensive mitigation strategies that enhance processor designs and instruction set architectures, providing robust defenses against side-channel attacks and covert data leaks [63, 70, 7, 64, 91]. By leveraging innovative methods, it is possible to secure speculative execution paths without sacrificing performance gains, thus protecting modern computing systems against sophisticated speculation-based threats.

8.2 Detection and Mitigation Techniques

Detecting and mitigating risks associated with speculative execution require innovative strategies addressing vulnerabilities in modern microarchitectural designs. Guarnieri et al. propose a framework for comparing security guarantees across various secure speculation mechanisms, emphasizing adaptable security measures for evolving threats [81]. SafeSpec introduces a novel approach by storing speculative state separately, minimizing data leak potential through speculative side channels [71]. MuonTrap employs a speculative filter cache to prevent sensitive information leakage, showcasing the efficacy of hardware-based solutions [67].

The SPECTECTOR algorithm offers a principled method for detecting speculative non-interference, providing automatic security proofs or identifying violations, thus serving as a robust tool for evaluating countermeasures against speculative execution attacks [92]. BranchSpec proposes mitigation techniques such as restoring branch predictor unit states after mis-speculation and delaying updates to the pattern history table, enhancing speculative execution security by addressing vulnerabilities in branch prediction mechanisms [93].

Maisuradze and Rossow emphasize the need to understand and mitigate security implications of speculative execution, proposing countermeasures against identified vulnerabilities [63]. Their research underscores the importance of ongoing investigation into hardware-level defenses to effectively address speculative execution risks. The experimental setup by Ponce de León et al. suggests a modular approach to defining speculative semantics, enabling rapid adaptation to new attack types [94].

SpecControl introduces a communication interface allowing compilers and application developers to inform hardware about true branch dependencies and confidential control-flow instructions, facilitating precise control over speculative execution paths without significant performance trade-offs [80].

The detection and mitigation of speculative execution risks require a comprehensive strategy integrating advanced techniques, thorough assessments, and continuous research into hardware and software defenses. This approach is especially vital in light of vulnerabilities exposed by attacks such as Spectre and Meltdown, which exploit speculative execution to leak sensitive data through side channels. Recent studies indicate that speculative execution can not only exfiltrate data but also conceal malicious computations from detection mechanisms, emphasizing the need for innovative defenses capable of addressing both exploitation and potential malware concealment within speculative execution contexts [63, 91]. By tackling the diverse challenges posed by speculative execution, these strategies aim to secure modern computing systems against sophisticated attack vectors while preserving performance.

8.3 Innovative Approaches and Future Directions

Innovative approaches in speculative execution are evolving, focusing on enhancing performance while addressing security concerns. The development of frameworks like SPECTECTOR highlights the importance of adaptable detection mechanisms that can respond to the dynamic nature of speculative execution threats. Future research could enhance SPECTECTOR to support a broader range of instructions and tackle path explosion issues in symbolic execution, indicating promising future directions [92].

The CAT framework represents a significant innovation in modeling speculative execution, providing a unified and flexible approach for rapid adaptation to new attacks while facilitating effective defense validation. Future research directions may include enhancing tool performance, exploring non-interference properties, and refining axiomatic models to accommodate a broader range of speculative execution behaviors [94]. SpecControl introduces a hardware/software co-design methodology that effectively protects against current and future speculation-based attacks, significantly reducing performance overhead while offering stronger security guarantees [80]. Future research could further optimize this co-design approach to enhance both security and performance.

The PROSPECT model presents another promising direction, with future research focusing on automating the annotation process for secret data in programs and optimizing the model for improved performance while ensuring security [6]. This could lead to more robust and efficient speculative execution mechanisms that uphold high security standards. Exploring additional metrics for vulnerability detection and applying Specure to a broader range of processor architectures could refine existing methods, enhancing their applicability and effectiveness in detecting speculative execution vulnerabilities [26]. Furthermore, improving input generation techniques and exploring white-box analysis of CPUs are promising research directions that could enhance testing coverage against complex speculative vulnerabilities [55].

The SPEECHMINER framework suggests that future research may focus on extending its capabilities to analyze other SPEECH variants and exploring the impacts of hardware extensions on vulnerability assessment, indicating innovative approaches and potential future directions [68]. Future research should prioritize developing robust mitigation strategies that balance security and performance while exploring the implications of emerging processor technologies on transient execution vulnerabilities [14]. This includes optimizing transformations for better performance and evaluating methods against emerging attack techniques [5]. Addressing these areas will contribute to the development of more secure and efficient computing systems that leverage the full potential of speculative execution.

9 Conclusion

Data prefetching and speculative execution are integral to enhancing hardware performance by addressing memory latency and boosting processor throughput. The DSMT architecture exemplifies the potential of these techniques, achieving significant performance gains, yet challenges like misspeculation persist. Prefetching strategies, such as MLOP, demonstrate their value by substantially improving system performance and cache efficiency. Truffle, for instance, significantly reduces execution times in serverless environments, showcasing its effectiveness in data-intensive workflows.

Selecting appropriate prefetching techniques tailored to specific workload characteristics is crucial, as highlighted by Bakhshalipour et al. Optimization opportunities, such as segment size selection, can further enhance cache performance in graph applications. The Redundant-small policy, as demonstrated by Aktas et al., offers a practical solution for mitigating stragglers, performing on par with more complex methods.

Despite advancements, speculative execution introduces security concerns, with vulnerabilities like those identified by KLEESPECTRE necessitating robust defenses. Frameworks such as Chronos illustrate the potential for improving speculative execution performance while ensuring security. Colvin's research highlights the importance of capturing speculative execution side effects for effective vulnerability analysis.

Future research should focus on developing comprehensive defenses against speculative execution vulnerabilities and exploring adaptive algorithms and machine learning techniques for broader hardware optimization applications. Addressing speculative execution challenges and enhancing large language model performance remain promising areas for exploration. Additionally, advancing prefetching strategies through innovative learning algorithms and integration with other hardware techniques could lead to significant performance improvements.

References

[1] Daniel Ortiz-Arroyo and Ben Lee. Dynamic simultaneous multithreaded architecture, 2024.

[2] Cynthia Marcelino and Stefan Nastic. Truffle: Efficient data passing for data-intensive serverless workflows in the edge-cloud continuum, 2024.

[3] Mehran Shakerinava, Mohammad Bakhshalipour, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Multi-lookahead offset prefetching. *The Third Data Prefetching Championship*, (2019), 2019.

[4] Chen Zhang, Zhuorui Liu, and Dawei Song. Beyond the speculative game: A survey of speculative execution in large language models, 2024.

[5] Davide Davoli, Martin Avanzini, and Tamara Rezk. Comprehensive kernel safety in the spectre era: Mitigations and performance evaluation (extended version), 2024.

[6] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. Prospect: Provably secure speculation for the constant-time policy (extended version), 2023.

[7] Robert J. Colvin and Kirsten Winter. An abstract semantics of speculative execution for reasoning about security vulnerabilities, 2020.

[8] Vikram Saraph and Maurice Herlihy. An empirical study of speculative concurrency in ethereum smart contracts, 2019.

[9] Mohammad Bakhshalipour, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Domino temporal data prefetcher. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 131–142. IEEE, 2018.

[10] Jiajun Wang, Reena Panda, and Lizy Kurian John. Prefetching for cloud workloads: An analysis based on address patterns. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 163–172. IEEE, 2017.

[11] Shahriar Ebrahimi, Reza Salkhordeh, Seyed Ali Osia, Ali Taheri, Hamid Reza Rabiee, and Hossein Asadi. Rc-rnn: Reconfigurable cache architecture for storage systems using recurrent neural networks, 2021.

[12] Dingding Li, Mianxiong Dong, Yanting Yuan, Jiaxin Chen, Kaoru Ota, and Yong Tang. Seer-mcache: A prefetchable memory object caching system for iot real-time data processing. *IEEE Internet of Things Journal*, 5(5):3648–3660, 2018.

[13] Surim Oh, Mingsheng Xu, Tanvir Ahmed Khan, Baris Kasikci, and Heiner Litz. Udp: Utility-driven fetch directed instruction prefetching. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1188–1201. IEEE, 2024.

[14] Wenjie Xiong and Jakub Szefer. Survey of transient execution attacks, 2020.

[15] New models for understanding and.

[16] Oleksii Oleksenko, Marco Guarnieri, Boris Köpf, and Mark Silberstein. Hide and seek with spectres: Efficient discovery of speculative information leaks with random testing, 2023.

[17] Mohammad Bakhshalipour, Mehran Shakerinava, Fatemeh Golshan, Ali Ansari, Pejman Lotfi-Karman, and Hamid Sarbazi-Azad. A survey on recent hardware data prefetching approaches with an emphasis on servers. *arXiv preprint arXiv:2009.00715*, 2020.

[18] Maotong Xu, Sultan Alamro, Tian Lan, and Suresh Subramaniam. Chronos: A unifying optimization framework for speculative execution of deadline-critical mapreduce jobs, 2018.

[19] Mehmet Fatih Aktas and Emina Soljanin. Optimizing redundancy levels in master-worker compute clusters for straggler mitigation, 2019.

[20] Chien-Hung Chen, Ting-Yuan Hsia, Yennun Huang, and Sy-Yen Kuo. Scheduling-aware data prefetching for data processing services in cloud. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 835–842. IEEE, 2017.

[21] Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. Hfetch: Hierarchical data prefetching for scientific workflows in multi-tiered storage environments. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 62–72. IEEE, 2020.

[22] Pengmiao Zhang, Rajgopal Kannan, Ajitesh Srivastava, Anant V Nori, and Viktor K Prasanna. Resemble: reinforced ensemble framework for data prefetching. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2022.

[23] Eshan Bhatia, Gino Chacon, Seth Pugsley, Elvira Teran, Paul V Gratz, and Daniel A Jiménez. Perceptron-based prefetch filtering. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 1–13, 2019.

[24] Haoyuan Wang and Zhiwei Luo. Data cache prefetching with perceptron learning. *arXiv preprint arXiv:1712.00905*, 2017.

[25] Xaver Fabian, Marco Guarnieri, and Marco Patrignani. Automatic detection of speculative execution combinations, 2022.

[26] Mohamadreza Rostami, Shaza Zeitouni, Rahul Kande, Chen Chen, Pouya Mahmoody, Jeyavijayan, Rajendran, and Ahmad-Reza Sadeghi. Lost and found in speculation: Hybrid speculative vulnerability detection, 2024.

[27] Exploiting vector code semantics.

[28] Yuxuan Zhang, Nathan Sobotka, Soyoon Park, Saba Jamilan, Tanvir Ahmed Khan, Baris Kasikci, Gilles A Pokam, Heiner Litz, and Joseph Devietti. Rpg2: Robust profile-guided runtime prefetch generation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 999–1013, 2024.

[29] Majid Jalili and Mattan Erez. Reducing load latency with cache level prediction, 2021.

[30] Nathan Beckmann, Haoxian Chen, and Asaf Cidon. {LHD}: Improving cache hit rate by maximizing hit density. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 389–403, 2018.

[31] K. F. D. Rietveld and H. A. G. Wijshoff. Redefining the query optimization process, 2022.

[32] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Compute caches. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–492. IEEE, 2017.

[33] Yuhao Ju and Jie Gu. A 65nm systolic neural cpu processor for combined deep learning and general-purpose computing with 95% pe utilization, high data locality and enhanced end-to-end performance. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022.

[34] Ashish Shrivastava, Alan Gatherer, Tong Sun, Sushma Wokhlu, and Alex Chandra. Slap: A split latency adaptive vliw pipeline architecture which enables on-the-fly variable simd vector-length, 2021.

[35] Yunming Zhang, Vladimir Kiriansky, Charith Mendis, Saman Amarasinghe, and Matei Zaharia. Making caches work for graph analytics. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 293–302. IEEE, 2017.

[36] Karim Elsayed and Amr Rizk. On the impact of network delays on time-to-live caching, 2022.

[37] Priyank Faldu. Addressing variability in reuse prediction for last-level caches, 2020.

[38] Philipp Schmitz, Tobias Jauch, Alex Wezel, Mohammad R. Fadiheh, Thore Tiemann, Jonah Heller, Thomas Eisenbarth, Dominik Stoffel, and Wolfgang Kunz. Okapi: Efficiently safeguarding speculative data accesses in sandboxed environments, 2024.

[39] Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. *arXiv preprint arXiv:2203.16487*, 2022.

[40] Biswabandan Panda. Clip: Load criticality based data prefetching for bandwidth-constrained many-core systems. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 714–727, 2023.

[41] Abhishek Bhattacharjee. Translation-triggered prefetching. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63–76, 2017.

[42] Samuel Pakalapati and Biswabandan Panda. Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 118–131. IEEE, 2020.

[43] Meng Wu and Chao Wang. Abstract interpretation under speculative execution. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 802–815, 2019.

[44] Sizhuo Zhang, Arvind, and Muralidaran Vijayaraghavan. Taming weak memory models, 2016.

[45] Haneen Mohammed. Continuous prefetch for interactive data applications. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2841–2843, 2020.

[46] Jari Pronold, Jakob Jordan, Brian J. N. Wylie, Itaru Kitayama, Markus Diesmann, and Susanne Kunkel. Routing brain traffic through the von neumann bottleneck: Efficient cache usage in spiking neural network simulation code on general purpose computers, 2022.

[47] Xinjian Long, Xiangyang Gong, Bo Zhang, and Huiyang Zhou. Deep learning based data prefetching in cpu-gpu unified virtual memory. *Journal of Parallel and Distributed Computing*, 174:19–31, 2023.

[48] Mengzhao Zhang, Qian Tang, Jeong-Geun Kim, Bernd Burgstaller, and Shin-Dug Kim. Adaptive regression prefetching algorithm by using big data application characteristics. *Applied Sciences*, 13(7):4436, 2023.

[49] Wim Heirman, Kristof Du Bois, Yves Vandriessche, Stijn Eyerman, and Ibrahim Hur. Near-side prefetch throttling: Adaptive prefetching for high-performance many-core processors. In *Proceedings of the 27th international conference on parallel architectures and compilation techniques*, pages 1–11, 2018.

[50] Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices. *Advances in Neural Information Processing Systems*, 37:16342–16368, 2024.

[51] Adita Kulkarni and Anand Seetharam. Model and machine learning based caching and routing algorithms for cache-enabled networks, 2020.

[52] Hao Wu, Krishnendra Nathella, Joseph Pusdesris, Dam Sunwoo, Akanksha Jain, and Calvin Lin. Temporal prefetching without the off-chip metadata. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 996–1008, 2019.

[53] Stefano Traverso, Mohamed Ahmed, Michele Garetto, Paolo Giaccone, Emilio Leonardi, and Saverio Niccolini. Unravelling the impact of temporal and geographical locality in content caching systems, 2015.

[54] Huanle Xu and Wing Cheong Lau. Optimization for speculative execution of multiple jobs in a mapreduce-like cluster, 2015.

[55] Oleksii Oleksenko, Christof Fetzer, Boris Köpf, and Mark Silberstein. Revizor: Testing black-box cpus against speculation contracts. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 226–239, 2022.

[56] Daniel Gerzhoy and Donald Yeung. *Pipelined CPU-GPU Scheduling for Caches*. 2021.

[57] Riyadh Baghdadi, Albert Cohen, Cedric Bastoul, Louis-Noel Pouchet, and Lawrence Rauchwerger. The potential of synergistic static, dynamic and speculative loop nest optimizations for automatic parallelization, 2011.

[58] Zirui Neil Zhao, Houxiang Ji, Mengjia Yan, Jiyong Yu, Christopher W Fletcher, Adam Morrison, Darko Marinov, and Josep Torrellas. Speculation invariance (invarspec): Faster safe execution through program analysis. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1138–1152. IEEE, 2020.

[59] Christine Fricker, Philippe Robert, and James Roberts. A versatile and accurate approximation for lru cache performance, 2012.

[60] Branden Butler, Sixing Yu, Arya Mazaheri, and Ali Jannesari. Pipeinfer: Accelerating llm inference using asynchronous pipelined speculation, 2024.

[61] Michele Garetto, Emilio Leonardi, and Stefano Traverso. Efficient analysis of caching strategies under dynamic content popularity, 2014.

[62] Giorgi Maisuradze and Christian Rossow. ret2spec: Speculative execution using return stack buffers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2109–2122, 2018.

[63] Giorgi Maisuradze and Christian Rossow. Speculose: Analyzing the security implications of speculative execution in cpus, 2018.

[64] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution, 2018.

[65] Uc berkeley.

[66] Shengjian Guo, Yueqi Chen, Peng Li, Yueqiang Cheng, Huibo Wang, Meng Wu, and Zhiqiang Zuo. Specusym: Speculative symbolic execution for cache timing leak detection. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1235–1247, 2020.

[67] Sam Ainsworth and Timothy M Jones. Muontrap: Preventing cross-domain spectre-like attacks by capturing speculative state. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 132–144. IEEE, 2020.

[68] Yuan Xiao, Yinqian Zhang, and Radu Teodorescu. Speechminer: A framework for investigating and measuring speculative execution vulnerabilities, 2019.

[69] Fabian Sperrle, Jürgen Bernard, Michael Sedlmair, Daniel Keim, and Mennatallah El-Assady. Speculative execution for guided visual analytics, 2019.

[70] Ofir Weisse, Ian Neal, Kevin Loughlin, Thomas F Wenisch, and Baris Kasikci. Nda: Preventing speculative execution attacks at their source. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 572–586, 2019.

[71] Khaled N Khasawneh, Esmaeil Mohammadian Koruyeh, Chengyu Song, Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Safespec: Banishing the spectre of a meltdown with leakage-free speculation. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[72] Pece Mitrevski and Marjan Gusev. On the performance potential of speculative execution based on branch and value prediction, 2013.

[73] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. Access-time-aware cache algorithms. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(4):1–29, 2017.

[74] Omer Irshad, Muhammad Usman Ghani Khan, Razi Iqbal, Shakila Basheer, and Ali Kashif Bashir. Performance optimization of iot based biological systems using deep learning. *Computer Communications*, 155:24–31, 2020.

[75] Minsik Cho, Mohammad Rastegari, and Devang Naik. Kv-runahead: Scalable causal llm inference by parallel key-value cache generation. *arXiv preprint arXiv:2405.05329*, 2024.

[76] Robert J. Colvin. Parallelized sequential composition, pipelines, and hardware weak memory models, 2021.

[77] Ruslan Svirschevski, Avner May, Zhuoming Chen, Beidi Chen, Zhihao Jia, and Max Ryabinin. Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices, 2024.

[78] Meng Wu and Chao Wang. Abstract interpretation under speculative execution, 2019.

[79] Jongouk Choi, Qingrui Liu, and Changhee Jung. Compiler directed speculative intermittent computation, 2020.

[80] Ali Hajiabadi, Archit Agarwal, Andreas Diavastos, and Trevor E. Carlson. Mitigating speculation-based attacks through configurable hardware/software co-design, 2023.

[81] Marco Guarnieri, Boris Köpf, Jan Reineke, and Pepe Vila. Hardware-software contracts for secure speculation. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1868–1883. IEEE, 2021.

[82] Rutvik Choudhary, Alan Wang, Zirui Neil Zhao, Adam Morrison, and Christopher W. Fletcher. Declassiflow: A static analysis for modeling non-speculative knowledge to relax speculative execution security measures (full version), 2023.

[83] Conor Green, Cole Nelson, Mithuna Thottethodi, and T. N. Vijaykumar. Safebet: Secure, simple, and fast speculative execution, 2023.

[84] Canberk Morelli and Jan Reineke. Warping cache simulation of polyhedral programs, 2022.

[85] Cristobal Ortega, Victor Garcia, Miquel Moreto, Marc Casas, and Roxana Rusitoru. Data prefetching on in-order processors. In *2018 International Conference on High Performance Computing & Simulation (HPCS)*, pages 322–329. IEEE, 2018.

[86] Pawan Poojary, Sharayu Moharir, and Krishna Jagannathan. Caching under content freshness constraints, 2017.

[87] Debabala Swain, Bijay Paikaray, and Debabrata Swain. Awrp: Adaptive weight ranking policy for improving cache performance, 2011.

[88] Vignesh Balaji, Neal Crago, Aamer Jaleel, and Brandon Lucia. P-opt: Practical optimal cache replacement for graph analytics. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 668–681. IEEE, 2021.

[89] Xiaoyang Lu, Hamed Najafi, Jason Liu, and Xian-He Sun. Chrome: Concurrency-aware holistic cache management framework with online reinforcement learning. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1154–1167. IEEE, 2024.

[90] Shengjian Guo, Yueqi Chen, Peng Li, Yueqiang Cheng, Huibo Wang, Meng Wu, and Zhiqiang Zuo. Specusym: Speculative symbolic execution for cache timing leak detection, 2020.

[91] Jack Wampler, Ian Martiny, and Eric Wustrow. Exspectre: Hiding malware in speculative execution. In *NDSS*, 2019.

[92] Marco Guarnieri, Boris Köpf, José F. Morales, Jan Reineke, and Andrés Sánchez. Spectector: Principled detection of speculative information flows, 2019.

[93] Md Hafizul Islam Chowdhuryy, Hang Liu, and Fan Yao. Branchspec: Information leakage attacks exploiting speculative branch instruction executions. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 529–536. IEEE, 2020.

[94] Hernán Ponce-de León and Johannes Kinder. Cats vs. spectre: An axiomatic approach to modeling speculative execution attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 235–248. IEEE, 2022.