Automated Graph Optimization for Cryptographic Neural Network Inference

Victor Manuel Sabater
Department of Computer Science, Colorado State University, Fort Collins, CO, USA.
victormanuelsabater05@colostate.edu

Madhavi L. Ramanathan
Department of Computer Science, Binghamton University, Binghamton, NY, USA.
madhavi.work@binghamton.edu

Rhianne MacCool
Department of Computer Science, University of New Hampshire, Durham, NH, USA.
maccool1995@unh.edu

April D. Kerr
School of Information Technology, University of Cincinnati, Cincinnati, OH, USA.
aprilmail@uc.edu

Abstract

Automated graph optimization for cryptographic neural network inference has emerged as a critical systems challenge at the intersection of privacy-enhancing cryptography, machine learning compilers, and distributed infrastructure. This paper examines how compiler-level transformations can reconcile the competing demands of cryptographic correctness, latency, throughput, memory pressure, energy consumption, and deployment governance. It argues that graph optimization for encrypted inference cannot be treated as a narrow arithmetic rewrite problem; rather, it is a socio-technical infrastructure problem in which optimization choices redistribute trust, cost, risk, and accountability across cloud providers, model owners, data subjects, regulators, and hardware vendors. The discussion covers homomorphic encryption, secure multiparty computation, hybrid protocols, and hardware acceleration, with attention to architectural trade-offs in optimization pipelines, verification and robustness, sustainability, fairness, and policy. Case illustrations from private inference services and edge deployments show that automated graph optimization can improve feasibility, but only when paired with transparent governance, reproducible benchmarking, and security validation. The paper concludes with a research agenda for composable, auditable, and sustainable cryptographic inference graphs.

Keywords

cryptographic neural network inference, graph optimization, homomorphic encryption, secure multiparty computation, privacy-enhancing technologies, systems architecture, governance, sustainability.

1. Introduction

Cryptographic neural network inference refers to the execution of machine learning models over encrypted or secretly shared inputs so that no single party necessarily observes both the model and the data in plaintext. The idea has moved from theoretical curiosity to practical

systems research because of advances in fully homomorphic encryption, secure multiparty computation, and hybrid protocols that combine several cryptographic primitives. Early fully homomorphic encryption established the possibility of computing arbitrary functions over encrypted data, even though its overhead was initially prohibitive for realistic neural networks [1]. Subsequent work demonstrated that neural network inference could be expressed as a sequence of encrypted operations with acceptable accuracy in limited settings [2]. Later improvements in lattice-based and torus-based encryption made encrypted computation more responsive, especially for Boolean and low-precision arithmetic [3]. Compiler frameworks then began to translate high-level neural network descriptions into cryptographic circuits and encrypted tensor programs, revealing that the main obstacle is not only cryptographic hardness but also graph-level inefficiency [4]. Automated graph optimization therefore becomes central: it determines which operations are fused, which nonlinearities are approximated, which data layouts are selected, and which cryptographic backends are invoked.

The importance of graph optimization is easy to underestimate when cryptographic inference is viewed as a single primitive. In practice, a neural network graph contains many structural opportunities for improvement, including elimination of redundant operations, algebraic simplification, constant folding, layout transformation, operation fusion, and replacement of expensive nonlinear functions. Cryptographic execution amplifies the value of such transformations because every plaintext multiplication, rotation, or communication round can be orders of magnitude more expensive than its conventional counterpart. A graph that is merely correct in plaintext may be unusable under encryption if it multiplies ciphertext depth, triggers excessive bootstrapping, or forces repeated communication between parties. Automated optimization thus acts as a bridge between machine learning semantics and cryptographic cost models. It must preserve model accuracy while reducing the number of expensive operations, the depth of computation, the volume of ciphertext interaction, and the amount of memory consumed by intermediate values.

This paper develops a system-level account of automated graph optimization for cryptographic neural network inference. It emphasizes architecture, trade-offs, deployment, robustness, sustainability, fairness, and policy rather than a single algorithmic trick. The central argument is that automated graph optimization should be understood as an infrastructural practice. Optimization decisions are not neutral technical choices; they shape who can deploy private inference, at what cost, under what assurances, and with what environmental footprint. A private inference service that runs efficiently on a large cloud cluster may remain inaccessible to small clinics, public-interest researchers, or edge deployments. A graph transformation that improves latency may weaken interpretability or complicate verification. A caching strategy that improves throughput may create side-channel risks. These tensions require an interdisciplinary perspective that treats cryptographic inference as a socio-technical system, not only as a cryptographic or compiler problem.

2. Cryptographic Inference as a Systems Problem

Secure inference systems must satisfy a demanding combination of functional and nonfunctional requirements. They must preserve model accuracy, protect input confidentiality, protect model weights where required, resist adversarial manipulation, and produce results within service-level expectations. Early secure neural network frameworks showed that these goals can be partially reconciled through tailored protocols for matrix multiplication, convolutions, and activation functions [5]. Transformations of neural network operations into

oblivious primitives made it possible to hide access patterns while still computing predictions [6]. Scalable privacy-preserving machine learning systems further demonstrated that secure computation can support training and inference workloads when protocols are organized around efficient arithmetic and communication patterns [7]. Hybrid frameworks then combined homomorphic encryption, garbled circuits, and secret sharing to exploit the strengths of each primitive for different layers [8]. Service-oriented systems introduced practical cryptographic inference pipelines with attention to latency, model protection, and client-side preprocessing [9]. Two-party inference protocols also advanced the practical performance of private prediction by optimizing nonlinear operations and communication rounds [10]. Together, these systems show that cryptographic inference is not a monolithic technology but an assembly of protocol choices, data flows, and deployment assumptions.

Graph optimization sits above these protocols and must reason about their heterogeneous costs. A homomorphic encryption backend may favor deep arithmetic circuits because ciphertext packing and rotations are expensive but can be amortized across many values. A secure multiparty computation backend may favor low-depth arithmetic but tolerate more communication, provided that round complexity remains modest. A hybrid backend may split the network so that linear layers are evaluated under homomorphic encryption while nonlinear layers are evaluated through garbled circuits or secret sharing. Automated graph optimization must therefore discover partitions, rewrites, and schedules that respect the cost model of each backend. This is more than a matter of choosing the fastest kernel. It requires a representation of cryptographic depth, noise growth, packing efficiency, communication volume, memory residency, and trust assumptions. A transformation that is desirable in one backend may be harmful in another. For example, replacing a nonlinear activation with a polynomial approximation may reduce communication in a multiparty setting but increase multiplicative depth in a homomorphic setting. Conversely, a piecewise approximation may reduce depth but require comparison operations that are expensive under encryption.

The systems problem is also organizational. Cryptographic inference involves parties with different incentives. A model owner may want to protect proprietary weights. A client may want to protect sensitive input data. A cloud provider may want to minimize infrastructure cost. A regulator may demand auditability and data minimization. An end user may care about response time and battery consumption. Automated graph optimization makes choices that affect all of these actors, often without their direct involvement. If the optimizer selects a more aggressive approximation, it may reduce latency but alter the accuracy profile in ways that are difficult for a clinician or loan officer to explain. If it selects a particular hardware accelerator, it may improve performance but concentrate deployment capacity in the hands of vendors with specialized supply chains. If it caches encrypted intermediate results, it may improve throughput but raise questions about whether those results can be correlated across sessions. A purely technical optimizer that ignores these dimensions can produce technically efficient but socially fragile systems.

3. Graph Optimization across Arithmetic and Cryptographic Backends

A useful starting point for automated graph optimization is the observation that cryptographic inference graphs contain both machine learning structure and cryptographic structure. Machine learning structure includes layer types, tensor shapes, activation functions, normalization, pooling, and residual connections. Cryptographic structure includes ciphertext packing, rotation keys, noise budgets, communication rounds, and secret-sharing representations. Compiler frameworks for secure machine learning have attempted to bridge

these structures by offering tensor abstractions and automatic conversion between plaintext and encrypted execution [11]. Some systems train or fine-tune networks specifically for encrypted data, replacing difficult nonlinearities with approximations that are more amenable to homomorphic evaluation [12]. Libraries for encrypted tensor operations provide reusable primitives that make it easier to prototype graph transformations and compare backend costs [13]. Graph compilers for homomorphic deep learning have shown that device placement, operation fusion, and layout selection can be automated across encrypted tensors [14]. Other work has explored tricks for accelerating encrypted prediction as a service, including precomputation and protocol-specific transformations [15]. More recent methods automate the replacement of rectified linear units with alternatives that reduce cryptographic cost while preserving accuracy, illustrating how activation-level rewrites can materially change private inference performance [16].

The design space of graph optimization can be organized around several recurring trade-offs. The first is depth versus width. Reducing multiplicative depth is often crucial because deep circuits accumulate noise and may require bootstrapping or additional communication. However, depth reduction may require wider intermediate representations, more parallel operations, or more complex packing. The second is approximation versus exactness. Many nonlinear functions are expensive under encryption, so optimizers may substitute polynomial, piecewise, or lookup-based approximations. These substitutions can preserve accuracy statistically but may not preserve formal equivalence, which matters in regulated domains. The third is fusion versus modularity. Fusing operations can reduce memory traffic and communication, but it can also reduce the ability to reuse optimized kernels, inspect intermediate values, or swap backend implementations. The fourth is static versus dynamic optimization. Static optimization can plan ahead using known tensor shapes and cryptographic parameters, while dynamic optimization can adapt to runtime conditions such as network congestion or hardware availability. Static plans are easier to verify but may be brittle; dynamic plans are more flexible but harder to audit.

Backend heterogeneity complicates these trade-offs. Homomorphic encryption is attractive because it allows computation without interaction after key setup, but it imposes noise management and limits on nonlinear operations. Secure multiparty computation distributes trust and supports rich operations, but it requires communication and careful handling of collusion assumptions. Hybrid protocols combine both approaches, but they introduce conversion costs between representations. Automated graph optimization must therefore solve a partitioning problem that is simultaneously algebraic, cryptographic, and infrastructural. It must determine which subgraphs should be evaluated under which protocol, how data should be converted between representations, and how to schedule operations to minimize end-to-end latency. In practice, the best partition may depend on batch size, network topology, hardware acceleration, and even legal jurisdiction. A graph optimized for a data center with high-bandwidth interconnects may be inappropriate for a mobile client on a constrained wireless link. An optimizer that exposes these parameters to system operators can support context-sensitive deployment rather than one-size-fits-all compilation.

4. Architectural Trade-offs in Automated Optimization Pipelines

An automated optimization pipeline for cryptographic inference typically includes representation, analysis, transformation, scheduling, and validation stages. Representation defines how the neural network and cryptographic operations are described. Analysis estimates costs such as ciphertext depth, communication rounds, memory footprint, and

expected energy use. Transformation applies rewrites such as fusion, elimination, approximation, and partitioning. Scheduling assigns operations to devices, threads, or protocol phases. Validation checks correctness, security assumptions, and accuracy constraints. The architecture of this pipeline matters because different stages may rely on different abstractions. A transformation pass may operate on a high-level graph, while a scheduling pass may require low-level knowledge of packing layouts and hardware kernels. Secure computation platforms have shown that GPU acceleration can significantly improve cryptographic workloads, but such acceleration also introduces new constraints on memory movement and kernel design [17]. Surveys of homomorphic encryption provide a broad view of scheme families and implementation considerations, reinforcing that no single backend or parameter set dominates all use cases [18]. Compiler research in conventional deep learning demonstrates the value of automated end-to-end optimization across graph and operator levels [19]. Secure multiparty computation literature similarly emphasizes that protocol composition and secret-sharing structures are architectural choices with security and performance consequences [20].

A central architectural question is where optimization authority should reside. One option is a centralized compiler controlled by a cloud provider. This can produce highly tuned graphs for a specific hardware and cryptographic stack, but it may reduce portability and transparency. Another option is a modular compiler with pluggable backends, where model owners can select cryptographic providers, hardware targets, and validation regimes. Modularity improves contestability and avoids lock-in, but it can fragment optimization knowledge and complicate end-to-end guarantees. A third option is client-side or edge-side optimization, where the party closest to the data controls some transformations. This can improve privacy and latency, but it may exceed local compute budgets. The choice affects governance: centralized optimization can enforce consistent security policies, while decentralized optimization can better accommodate local legal and ethical requirements. A mature ecosystem may require both, with standard intermediate representations and auditable transformation logs.

Another trade-off concerns observability versus confidentiality. Optimizers need profiling data to improve performance, but profiling cryptographic inference can reveal sensitive patterns. For example, timing measurements may leak information about input sparsity, and memory access traces may reveal which branches were taken. An automated optimizer that collects telemetry from encrypted executions must therefore apply privacy-preserving measurement, aggregation, and retention policies. It may also need to separate performance telemetry from cryptographic secrets and to ensure that optimization caches do not become side channels. The architecture should support reproducibility without requiring disclosure of model weights or client data. This is not merely an engineering challenge; it is a governance requirement for systems that claim to provide private inference. Without it, optimization can undermine the very confidentiality that cryptographic protocols are intended to protect.

5. Robustness, Security, and Verification

Cryptographic inference graphs must be robust to both classical and cryptographic adversaries. Classical adversaries may attempt to manipulate inputs, poison models, or exploit distribution shift. Cryptographic adversaries may attempt to infer secrets from protocol transcripts, exploit incorrect parameter choices, or induce faults that reveal information. Automated graph optimization interacts with both threat models. If the optimizer replaces an activation with an approximation, it may change the model's adversarial robustness. If it fuses operations in a

way that depends on data values, it may introduce branching that leaks information. If it selects a weaker cryptographic parameter set to improve latency, it may reduce the security margin. Verification therefore cannot be an afterthought. It must be integrated into the optimization pipeline as a set of constraints that transformations must satisfy. These constraints should include semantic equivalence where required, bounded accuracy loss where approximation is allowed, cryptographic parameter validation, and side-channel analysis for data-dependent behavior.

Formal and semi-formal verification methods can help, but they must be practical. Full formal verification of large neural networks and cryptographic protocols remains difficult. A more feasible approach is layered assurance. At the graph level, optimizers can preserve a provenance record that documents each transformation and its justification. At the protocol level, implementations can be checked against known test vectors and security proofs. At the deployment level, runtime monitors can detect anomalous behavior, such as unexpected noise growth or communication patterns. The automatic replacement of expensive nonlinearities with cheaper alternatives illustrates why layered assurance is necessary [16]. Such transformations can be validated empirically on benchmark datasets, but empirical validation does not prove that accuracy will hold for all inputs. In safety-critical or rights-affecting applications, the optimizer may need to restrict aggressive rewrites or require human review. The same logic applies to hybrid protocol partitions, where changes in trust assumptions must be explicit rather than hidden in compiler heuristics.

Robustness also includes resilience to infrastructure failures. Cryptographic inference may involve multiple parties, remote procedure calls, hardware accelerators, and large encrypted tensors. A network partition or accelerator failure can interrupt a protocol and potentially expose intermediate state if recovery is not designed carefully. Automated graph optimization should therefore consider fault tolerance as a first-class objective. This may involve checkpointing encrypted intermediates, duplicating critical operations across parties, or designing protocols that can abort safely without revealing secrets. The optimizer should understand which operations are idempotent and which are not, and it should avoid schedules that concentrate risk in a single component. This is especially important for deployments in public-interest settings, where availability may matter as much as confidentiality. A private inference service that is secure but unreliable will not be adopted by hospitals, courts, or municipal agencies.

6. Deployment, Infrastructure, and Sustainability

Deployment context shapes the value of graph optimization. A cloud-based private inference service may prioritize throughput and cost per prediction, while an edge deployment may prioritize latency, battery life, and resistance to physical tampering. A cross-border deployment may need to respect data localization rules, which can constrain where ciphertexts, keys, and intermediate results are stored. Automated optimization can help by generating deployment-specific graphs, but it can also create a proliferation of variants that are difficult to maintain and certify. Infrastructure operators need tools for managing graph versions, cryptographic parameters, hardware requirements, and compliance evidence. They also need benchmarks that reflect real-world conditions rather than idealized microbenchmarks. Without such tools, optimization may improve one metric while degrading another, such as reducing latency at the cost of higher energy consumption or lower accuracy for underrepresented groups.

Sustainability is an increasingly important dimension of cryptographic inference. Encrypted computation is computationally intensive, and aggressive optimization may trade energy efficiency for speed or security. Homomorphic encryption can involve large ciphertexts and repeated polynomial operations, while secure multiparty computation can involve extensive communication. Hardware acceleration can improve energy efficiency per operation, but manufacturing and operating specialized accelerators also have environmental costs. Automated graph optimization can contribute to sustainability by reducing redundant computation, avoiding unnecessary bootstrapping, minimizing communication, and selecting energy-aware schedules. However, it can also encourage overprovisioning if performance is the only objective. A sustainable optimization pipeline should include energy and carbon metrics alongside latency and throughput. It should also consider the marginal value of privacy: some applications may justify higher energy use because they protect sensitive health or financial data, while others may not. These trade-offs should be visible to decision-makers rather than buried in compiler defaults.

Deployment also raises questions of equity. If optimized cryptographic inference requires specialized hardware, high-bandwidth connectivity, or expensive cloud credits, then its benefits may accrue mainly to well-resourced organizations. Smaller clinics, public defenders, community organizers, and researchers in low-resource settings may be excluded. Automated graph optimization can partly mitigate this by generating efficient graphs for commodity hardware and intermittent connectivity. It can also support progressive privacy, where less sensitive computations use lighter-weight protocols and more sensitive computations use stronger ones. Yet such flexibility must be governed carefully to avoid creating a two-tier system in which vulnerable populations receive weaker privacy protections. Infrastructure policy, public funding, and open-source tooling all matter here. The technical optimizer is necessary but not sufficient; it must be embedded in institutions that value equitable access.

7. Governance, Fairness, and Policy Implications

Governance of automated graph optimization must address accountability, transparency, and contestability. When an optimizer rewrites a model, who is responsible if the resulting predictions are biased or inaccurate? The model owner may blame the compiler, the compiler vendor may blame the cryptographic backend, and the backend may blame the deployment configuration. Clear responsibility chains are needed. This can be supported by transformation logs, versioned artifacts, and audit trails that record which optimizations were applied, why they were selected, and what validation was performed. Such records should be intelligible to regulators and affected parties, not only to engineers. They should also protect trade secrets and personal data, which requires careful separation of technical evidence from confidential model parameters and client inputs. The goal is not full disclosure but sufficient transparency to enable accountability.

Fairness is affected by graph optimization in subtle ways. Approximation errors are rarely uniform across input distributions. A polynomial approximation of an activation function may perform well on average but poorly on edge cases or minority subgroups. A quantization or pruning transformation may reduce accuracy more for rare classes. A partitioning decision may introduce latency that disproportionately affects users with slower networks. Automated optimizers that only optimize aggregate accuracy and average latency can therefore amplify existing inequities. Fairness-aware optimization would require evaluating transformations across relevant subgroups, documenting performance disparities, and allowing constraints that protect vulnerable populations. This is technically challenging because subgroup evaluation

may require sensitive data, but it is not impossible. Privacy-preserving evaluation methods, synthetic benchmarks, and third-party audits can help. The key point is that optimization objectives are normative choices. Selecting accuracy, latency, energy, and fairness weights is not a purely technical decision; it reflects institutional priorities.

Policy implications extend to standards, procurement, and international coordination. Governments and industry bodies may need standards for cryptographic parameters, benchmark reporting, and security validation. Procurement rules can require vendors to disclose optimization practices and provide audit access. Export controls and data protection laws may affect where cryptographic inference can be deployed and which optimizations are permissible. Because cryptographic inference often involves cross-border data flows, policy fragmentation can increase compliance costs and reduce interoperability. A coordinated approach could include common APIs, shared threat models, and mutual recognition of validation regimes. At the same time, policy should avoid freezing the technology too early. Cryptographic inference is evolving rapidly, and overly prescriptive rules may become obsolete or inhibit beneficial innovation. Adaptive governance, sandboxed deployments, and multistakeholder review can help balance stability with experimentation.

8. Case Illustrations and Cross-Domain Comparisons

Consider a hospital that wants to use a third-party diagnostic model without revealing patient images to the model owner. The hospital may encrypt images or secret-share them with a cloud provider. The model owner may encrypt or secret-share the weights. Automated graph optimization can reduce the cost of convolution, activation, and pooling layers, but it must also preserve diagnostic accuracy across diverse patient populations. If the optimizer approximates a nonlinearity too aggressively, it may miss subtle pathologies. If it partitions the network across jurisdictions, it may violate health data rules. If it caches intermediate encrypted representations, it may create new privacy risks. A successful deployment would combine performance optimization with clinical validation, data protection impact assessments, and human oversight. The optimizer is one component in a larger socio-technical arrangement.

Consider a financial institution that wants to detect fraud using a model trained by an external vendor. The institution may prioritize low latency because transactions must be scored in real time. The model vendor may prioritize intellectual property protection. A hybrid protocol might evaluate linear layers under homomorphic encryption and nonlinear layers through secure multiparty computation. Automated graph optimization could select fusion and partition strategies that minimize communication rounds. However, fairness concerns are acute: a transformation that improves average fraud detection may increase false positives for certain demographics. The institution would need subgroup evaluation, explainability, and appeal mechanisms. Policy requirements for credit and financial services may also demand documentation of model logic and validation. In this setting, optimization must be constrained by legal and ethical requirements, not only by speed.

Consider a public-interest organization that wants to offer private inference on low-cost edge devices for language translation or accessibility services. Here, energy and memory are scarce, and connectivity may be intermittent. Automated graph optimization might favor small models, aggressive quantization, and local execution over cloud offloading. Cryptographic strength may need to be balanced against battery life. The organization may accept a lower security margin for low-sensitivity tasks while requiring stronger protection for sensitive communications. This case illustrates that deployment context should determine optimization

objectives. A single global optimizer that prioritizes cloud throughput would be inappropriate. Instead, the system needs tunable policies, transparent trade-offs, and community participation in setting priorities.

9. Future Directions and Research Agenda

Future research should advance composable and auditable graph optimization. Composability means that transformations, protocol backends, and hardware targets can be combined without invalidating security or correctness assumptions. This requires formal interfaces between graph representations, cryptographic parameter sets, and runtime systems. Auditable optimization means that every significant transformation can be traced, explained, and validated. It also means that benchmarks are reproducible and that performance claims are tied to specific threat models and deployment conditions. Research on cryptographic compilers has already shown the value of end-to-end optimization, but more work is needed on cross-backend portability and verification [19]. Secure computation research provides a foundation for understanding protocol composition, but graph-level optimization for mixed protocols remains underexplored [20]. Advances in encrypted tensor libraries and approximate activation methods can improve performance, but they must be integrated with governance and validation frameworks [13] [16].

Another direction is adaptive optimization under uncertainty. Deployment conditions change: network bandwidth fluctuates, hardware accelerators fail, privacy regulations evolve, and adversarial threats shift. An adaptive optimizer could monitor these conditions and re-plan the graph at runtime, subject to safety constraints. This would require lightweight cost models, secure telemetry, and mechanisms for rolling back transformations that degrade accuracy or security. It would also require human oversight for high-stakes decisions. A fully autonomous optimizer may be inappropriate in domains such as healthcare or criminal justice, where accountability and contestability are essential. Instead, adaptive optimization could operate within bounded policy envelopes, with humans setting the acceptable trade-offs and reviewing exceptional cases.

Finally, research should address sustainability and equity as first-class objectives. Energy-aware compilation, carbon-aware scheduling, and hardware reuse can reduce the environmental footprint of cryptographic inference. Equitable optimization can ensure that privacy benefits are not limited to wealthy institutions. This may require public investment in open-source compilers, shared benchmarks, and training programs. It may also require policy interventions such as procurement standards, subsidies for public-interest deployments, and support for independent audits. The technical and social dimensions are intertwined. An optimization that is efficient but inaccessible, secure but unaccountable, or private but unfair will not fulfill the promise of cryptographic neural network inference.

10. Conclusion

Automated graph optimization is essential for making cryptographic neural network inference practical, but it is not merely a performance technique. It is a systems practice that mediates between machine learning semantics, cryptographic protocols, hardware infrastructure, and institutional governance. The most promising approaches combine graph-level rewrites with backend-aware cost models, hybrid protocol partitioning, and rigorous validation. They recognize that depth, communication, memory, energy, accuracy, and fairness are interdependent. They also recognize that optimization decisions distribute power and risk across parties. A private inference system that is fast but opaque, or secure but inaccessible,

may fail in the contexts where privacy matters most. The research agenda therefore must be interdisciplinary. It must develop composable and auditable optimizers, robust verification methods, sustainable deployment strategies, and governance frameworks that make trade-offs explicit. If these challenges are addressed, automated graph optimization can help realize cryptographic inference as a trustworthy, equitable, and durable component of socio-technical infrastructure.

References

1. Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In Proceedings of the 41st Annual ACM Symposium on Theory of Computing (pp. 169–178). Association for Computing Machinery.

2. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., & Wernsing, J. (2016). CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In Proceedings of the 33rd International Conference on Machine Learning (pp. 201–210). PMLR.

3. Chillotti, I., Gama, N., Georgieva, M., & Izabachène, M. (2020). TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology, 33(1), 34–91.

4. Dathathri, R., Saarikivi, O., Chen, H., Laine, K., Lauter, K., Maleki, S., & Musuvathi, M. (2019). CHET: An optimizing compiler for fully homomorphic neural-network inference. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (pp. 1–16). Association for Computing Machinery.

5. Juvekar, C., Vaikuntanathan, V., & Chandrakasan, A. (2018). GAZELLE: A low latency framework for secure neural network inference. In Proceedings of the 27th USENIX Security Symposium (pp. 1651–1669). USENIX Association.

6. Liu, J., Juuti, M., Lu, Y., & Asokan, N. (2017). Oblivious neural network predictions via MiniONN transformations. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 619–631). Association for Computing Machinery.

7. Mohassel, P., & Zhang, Y. (2017). SecureML: A system for scalable privacy-preserving machine learning. In 2017 IEEE Symposium on Security and Privacy (pp. 19–38). IEEE.

8. Riazi, M. S., Weinert, C., Tkachenko, O., Songhori, E. M., Schneider, T., & Koushanfar, F. (2018). Chameleon: A hybrid secure computation framework for machine learning applications. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security (pp. 707–721). Association for Computing Machinery.

9. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., & Popa, R. A. (2020). Delphi: A cryptographic inference service for neural networks. In Proceedings of the 29th USENIX Security Symposium (pp. 2505–2522). USENIX Association.

10. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., & Sharma, R. (2020). CrypTFlow2: Practical 2-party secure inference. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (pp. 325–342). Association for Computing Machinery.

11. Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., & van der Maaten, L. (2021). CrypTen: Secure multi-party computation meets machine learning. In Advances in Neural Information Processing Systems 34 (pp. 4961–4973). Curran Associates.

12. Lou, Q., & Jiang, L. (2019). SHE: A fast and accurate deep neural network for encrypted data. In Advances in Neural Information Processing Systems 32 (pp. 10057–10065). Curran Associates.

13. Benaissa, A., Retiat, B., Cebere, B., & Belfiore, J.-C. (2021). TenSEAL: A library for encrypted tensor operations using homomorphic encryption. arXiv. https://arxiv.org/abs/2104.03152

14. Boemer, F., Costache, A., Cammarota, R., & Wierzynski, C. (2019). nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data. In Proceedings of the 16th ACM International Conference on Computing Frontiers (pp. 3–13). Association for Computing Machinery.

15. Sanyal, A., Kusner, M. J., Gascón, A., & Kanade, V. (2018). TAPAS: Tricks to accelerate (encrypted) prediction as a service. In Proceedings of the 35th International Conference on Machine Learning (pp. 4490–4499). PMLR.

16. Peng, H., Huang, S., Zhou, T., Luo, Y., Wang, C., Wang, Z., ... & Ding, C. (2023, October). Autorep: Automatic relu replacement for fast private network inference. In 2023 IEEE/CVF International Conference on Computer Vision (ICCV) (pp. 5155-5165). IEEE.

17. Watson, J.-L., Wagh, S., & Popa, R. A. (2022). Piranha: A GPU platform for secure computation. In Proceedings of the 31st USENIX Security Symposium (pp. 827–844). USENIX Association.

18. Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys, 51(4), 1–35.

19. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., ... & Ceze, L. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (pp. 578–594). USENIX Association.

20. Cramer, R., Damgård, I., & Nielsen, J. B. (2015). Secure multiparty computation and secret sharing. Cambridge University Press.