CCBNet: Confidential Collaborative Bayesian Networks Inference

Abele Mălan[1,⋆][0009−0002−4493−7439], Thiago Guzella[2][0000−0002−1374−7379], Jérémie Decouchant[3][0000−0001−9143−3984], and Lydia Chen[1,3][0000−0002−4228−6735]

[1] University of Neuchâtel {abele.malan, yiyu.chen}@unine.ch
[2] ASML thiago.guzella@asml.com
[3] Delft University of Technology j.decouchant@tudelft.nl

Abstract. Effective large-scale process optimization in manufacturing industries requires close cooperation between different human expert parties who encode their knowledge of related domains as Bayesian network models. For instance, Bayesian networks for domains such as lithography equipment, processes, and auxiliary tools must be conjointly used to effectively identify process optimizations in the semiconductor industry. However, business confidentiality across domains hinders such collaboration, and encourages alternatives to centralized inference. We propose CCBNet, the first **C**onfidentiality-preserving **C**ollaborative **B**ayesian **Net**works inference framework. CCBNet leverages secret sharing to securely perform analysis on the combined knowledge of party models by joining two novel subprotocols: (i) CABN, which augments probability distributions for variables across parties by modeling them into secret shares of their normalized combination; and (ii) SAVE, which aggregates party inference result shares through distributed variable elimination. We extensively evaluate CCBNet via 9 public Bayesian networks. Our results show that CCBNet achieves predictive quality that is similar to the ones of centralized methods while preserving model confidentiality. We further demonstrate that CCBNet scales to challenging manufacturing use cases that involve 16–128 parties in large networks of 223–1003 variables, and decreases, on average, computational overhead by 23%, while communicating 71k values per request. Finally, we showcase possible attacks and mitigations for partially reconstructing party networks in the protocol.

1 Introduction

Improving productivity and quality standards in manufacturing demands effectively expressing complex interactions between domain items. Bayesian networks (BNs) are commonly adopted to graphically model causality in manufacturing [29], with nodes representing variables and directed edges showing dependencies. An essential trait of these models is their ability to specify arbitrary input and output variables for each query instead of having them fixed.

⋆ Work partly done while graduating from TU Delft and interning at ASML.

In the semiconductor industry, the pursuit of smaller chips at a high yield [39] entails cooperation among specialized parties protecting their trade secrets. More specifically, fab operators encode their insight into settings dictating the production process in a BN. Similarly, vendors of equipment like scanners, construct BNs that describe the inner workings of their machines. Pooling together parties' knowledge allows to better optimize production environments, leading to new business opportunities. Simultaneously, the need to protect intellectual property expressed within party models calls for confidential collaboration.

Existing studies on collaborative inference for BNs **ignore model confidentiality** or limit the ability to merge party information. We consider confidentiality preserved if parties cannot infer from others the identity and structural connections of new variables, or parameters of other probability tables, even for variables they also define themselves. Most previous works detail a centralized combination of local BNs into a larger one [13,16] without protecting confidential knowledge within the input networks and global output. Models get stitched together based on common nodes, and remain in the final representation as largely unaltered submodels whose encoded knowledge is easily inspectable. Pavlin et al. [31] propose a distributed combination variant that partially preserves confidentiality by maintaining the locality of combined party models. However, it leaks information when combining them and does not allow merging inner graph nodes having both parents and children. Tedesco et al. [37] maintain the confidentiality of how nodes are linked within parties but only allow propagating information between them in a fixed sequential order. The approach of Kim and Ghahramani [24] has similar confidentiality properties but even greater compatibility restrictions by requiring models to have identical inputs and outputs.

In this paper, we propose `CCBNet`, the first confidential, collaborative BNs inference framework that combines knowledge of multiple parties involved in inference queries through a novel secret sharing scheme. `CCBNet` does not require a trusted third party, and protects confidentiality at both the levels of party models and data instances. The two key components of `CCBNet` are: (i) confidential sharing of aggregated variable probability distributions across all overlapping parties; and (ii) distributed inference based on variable elimination for aggregating party results. The novelty of the augmentation procedure lies in constructing discrete conditional probability distributions for all variables present in more than one party. These variables represent secret shares of a combined and normalized distribution from a centralized scenario without exposing any party's initial probability function. To evaluate `CCBNet`, we simulate knowledge compartmentalization over different public BNs. Moreover, we demonstrate possible attacks against the framework, their limitations, and possible mitigations.

In summary, we make the following contributions:

- We devise a confidential, collaborative framework for BNs, `CCBNet`, to satisfy the needs of industry use cases like process optimization in manufacturing.
- We design a novel secret sharing-based protocol, `CABN`, to confidentially augment the conditional probability function of overlapping variables in parties.
- We define `SAVE`, a method backed by variable elimination for performing dis-

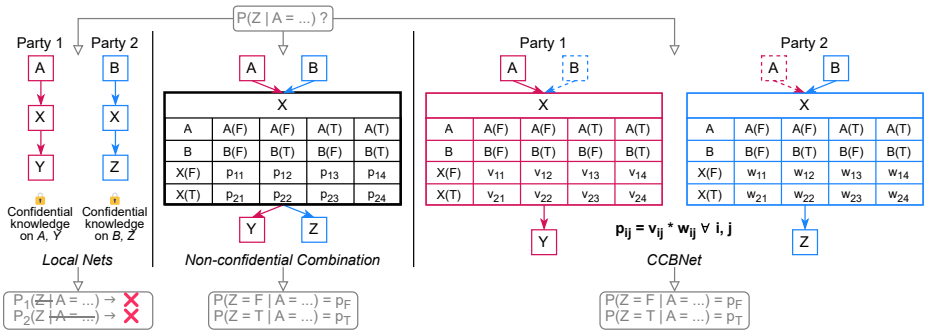


Fig. 1: **Bayesian networks collaboration comparison:** Variables are X (shared), A, B, Y, and Z. They all have two states, **F**alse and **T**rue. Entries p_{ij} are conditional probability values (e.g., $p_{11} = P(X = F \mid A = F,\ B = F)$). v_{ij} and w_{ij} are secret shares which multiplied reconstruct p_{ij}. After observing A, inferring Z's updated posterior $P(Z \mid A = ...)$ requires propagating information between A, B, and X, which is impossible by averaging model outputs. Meanwhile, the non-confidential approach reveals the combined graph and probability tables to parties. `CCBNet` exchanges minimal structural information among parties and runs distributed inference with secretly shared confidential values.

tributed secret sharing inference on augmented party models.
• We evaluate `CCBNet` over various scenarios based on nine public BNs.
Our results indicate that `CCBNet`'s predictive accuracy is similar to those of non-confidential centralized alternatives and, for many collaborators in large networks, that `CCBNet` decreases the computational overhead by 23% on average when 71k values are communicated per request.
• We discuss possible attacks and mitigations on the two `CCBNet` subprotocols.
Our code is available at: https://anonymous.4open.science/r/ccbnet-E861.

2 Background & Related Studies

2.1 Background

Bayesian Networks are probabilistic graphical models that maintain explicit conditional probability distributions (CPDs) for variables which form a directed acyclic graph [36]. Shown in Figure 1, variables and the influences between them give the graph nodes and edges, respectively. Principally for performance and interpretability, variables in practical applications are generally discrete, with CPDs specifically embodying conditional probability tables. Learning may be driven by data, human experts, or both [25,11], as with other human-readable models like decision trees. Automated learning discovers the graph structure and then populates CPD parameters from training data. Manual learning is desirable when incorporating concepts with known governing rules that need no approximation from observations.

Computation in discrete BNs relies on a few base operations for propagating information: normalization, reduction, marginalization, and products [25]. We outline them with help from the non-confidential combination in Figure 1. Normalizing a CPD divides its entries by their column sum. Thus column summations, like $p_{11} + p_{21}$, would become 1. Reduction and marginalization remove variables from a CPD by fixing their states or, respectively, summing them out. Reducing or marginalizing A from X leaves B as its sole parent. Previous operations apply to both representations. Products operate on CPDs of the same variable or factors and create a new CPD/factor over the input variables' union, where each entry is the multiplication of the corresponding ones in the original representations. The product of X and Z's factors, thus, also contains A and B.

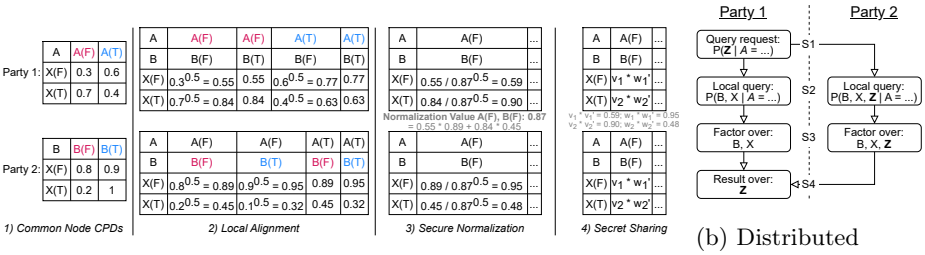
In the considered discrete scenario, a CPD with target variable X, and parents $A, B, ...$ is denoted as an $(S_X \times S_A \times S_B \times ...)$ array, accompanied by a mapping between dimensions to variables and indices to states. S_V is the number of states for variable V. Each entry specifies a probability value for the state combination it represents. The corresponding factor is identical but does not distinguish between a target and parents. Any continuous variables would need to have a globally agreed-upon discretization applied to them before usage.

Inference in BNs finds updated posterior probabilities for the states of target variables, given the observed states of any evidence variables [33,32]. As general inference is NP-Hard, approximate algorithms help decrease computation costs compared to exact ones while sacrificing some precision in the result. The main exact inference techniques are variable elimination (VE) and junction tree belief propagation, which decomposes the network into a tree of variable clusters, runs VE within them and then disseminates updates between neighbors by message-passing [25]. In Figure 1's query, Z is the target, given some observed state of A. The VE algorithm performs three main steps over the factors corresponding to the model's CPDs. It first reduces the evidence from input factors. Reduction merely discards entries in the factor where evidence variables have a different state than specified. Then, it iteratively performs products between factors and marginalizes them for each non-query variable. Lastly, it takes the product between the leftover factors to get the final result factor.

Markov random fields (MRFs) are a sibbling model of BNs, backed by undirected graphs, into which every BN is easily transformable via moralization [26,35]. Apart from lacking acyclicity constraints, MRFs directly define parameters as factors and can deal with scenarios where edge directionality is unspecified, but BNs are more compact and efficient for generative use. For inference, BN properties and algorithms remain applicable. Thus, when aciclicity constraints are unsatisfiable, MRFs can still perform inference like in BNs.

2.2 Prior Art

We identify two high-level categories of collaborative analysis for BNs: single- and multi-model. The first creates one global model, while the other keeps local models separate, merging their results.



(a) Key Steps of computing augmented CPDs

(b) Distributed inference flow

Fig. 2: CABN & SAVE steps for Figure 1 scenario

Single-model approaches harness party data instances or models but neglect model confidentiality. Federated learning discovers models [30,17,21,1,10] from private party instances with a coordinator. Fully decentralized formulations also exist [6,18]. Direct network combination fuses structure [13,2] and parameters [16] from party models.

Multi-model methods have parties work together during inference to produce a complete analysis result. Tedesco et al. [37] chain model without exchanging their contents but only allows using them one at a time in a predetermined order. Less confidential but more flexible, the work of Pavlin et al. [31] fuses party networks based on common nodes but still requires them to be roots or leaves in the party's directed acyclic graph. Ypma et al. [40] patent a collaborative solution for industrial processes, but only mention data confidentiality preservation by anonymization. Kim and Ghahramani [24] run models autonomously and only average their final outputs, maintaining confidentiality but expecting models to share inputs and outputs.

Single-model solutions break confidentiality by centralizing knowledge. Multi-model ones trade modeling power for confidentiality. CCBNet addresses both.

3 CCBNet

We propose, CCBNet, a framework for secure distributed analysis over a related set of confidential and discrete BNs. CCBNet is composed of two key steps, (i) CABN augments the BNs of parties through overlapping variables, and (ii) SAVE performs joint inference on them.

The assumptions we make are that variables from different parties have the same name only if they represent the same concept, and the independence, across parties, of distinct parents for the same node reasonably approximates the ground truth. These are shared by previous BN combination works. Thus, names identify the overlapping (common) nodes between models, giving the contact points for graph fusion. Since variables modeled by parties may be any subset of those from the full domain, modeling direct interactions between their non-overlapping variables requires great amounts of often unavailable information.

Algorithm 1 `CABN`

Input: $P = \{p_1,\ p_2,\ \dots,\ p_n\}$ – parties,
$CPD_p[T \mid A,\ B,\ \dots]$ – CPD in party p for target var T with parent vars $\{A,\ B,\ \dots\}$,
$weight_p$ – weight assigned by p to its BN (defaults to 1)
Output: updated $CPD_p[O \mid \dots]$, $\forall$ overlap var O modeled by party p

for each var V of parties from P **do**
  $parties_V \leftarrow \varnothing$ {list of parties containing V}
  $states_V \leftarrow \varnothing$ {list of overlap states for V}
for each pair of parties $(p,\ p')$ from P **do**
  for each var $O \in PSI$(vars of p, vars of p') **do**
    $parties_O \leftarrow parties_O \cup \{p,\ p'\}$
for each var O s.t. $parties_O \neq \varnothing$ **do**
  for each var set $\mathcal{V} = \{X, Y, \dots\}$ s.t. $(p \in parties_O \wedge CPD_p[O \mid \mathcal{V}]$ exists) **do**
    $states_V \leftarrow states_V\ \cup\ \{v_0,\ v_1,\ \dots\}$ {where v_i is a state of var V}
  $idCPD \leftarrow$ CPD for target var O and $states_V$ with all entries 1
  wSum $\leftarrow \sum_{p \in parties_O} weight_p$
  for each party $p \in parties_O$ **do**
    $CPD_p[O \mid \dots] \leftarrow CPD_p[O \mid \dots] \cdot idCPD$ {factor prod}
    $CPD_p[O \mid \dots] \leftarrow CPD_p[O \mid \dots]^{weight_p / wSum}$ {element-wise exp}
  for each parent config $\mathcal{S} = (X = x_i,\ Y = y_j, \dots)$ s.t. $idCPD[O \mid \mathcal{S}]$ exists **do**
    $prodHE \leftarrow \odot_{p \in parties_O} HE_{enc}(CPD_p[O \mid \mathcal{S}])$ {enc element-wise prod}
    $norm \leftarrow HE_{dec}(||prodHE||_1)^{1/|parties_O|}$ {dec normalization scalar}
    for each party $p \in parties_O$ **do**
      $CPD_p[O \mid \dots] \leftarrow CPD_p[O \mid \dots]\ /\ norm$
  for each party $p \in parties O$ **do**
    $CPD_p^0[O \mid \dots],\ CPD_p^1[O \mid \dots],\ \dots \leftarrow$ secret share $CPD_p[O \mid \dots]$
  for each party $p_i \in parties O$ **do**
    $CPD_{p_i}[O \mid \dots] \leftarrow \prod_{p \in parties_O} CPD_p^i[O \mid \dots]$
return $CPD_p[O \mid \dots]$ $, \forall\ parties_O \neq \varnothing,\ p \in parties_O$

Our adversarial model includes semi-honest parties that follow the protocol while trying to abuse gained information [19] but do not collude. No trusted third party exists. The goal is to protect all network parameters and only disclose structure/state-name information among parties modeling the same variables.

In Appendix A, we give more context regarding our method's flexibility and applicability. Specifically, we outline configuration tweaks for different security and performance tradeoffs and possible functionality extensions. We also discuss the relevance of the tackled adversarial model. We additionally include a comparison with confidential computing methods in Appendix B.

3.1 Confidentially Augmented Bayesian Networks

We now present the `CABN`[4] protocol, which updates local CPDs for overlap variables to hold secret shares of their combination, protecting the initial probabilities. Algorithm 1 details the four steps of the protocol, illustrated in

[4] **C**onfidentially **A**ugmented **B**ayesian **N**etworks

Figure 2a: (i) private common node identification; (ii) local alignment; (iii) secure normalization; and (iv) secret sharing. The protocol updates parties when the number of changes in local networks passes a set threshold.

Overview. Structurally, CABN imitates a union of the involved networks of Del Sagrado and Moral[13]. Parameter-wise, it follows Feng et al. [16] but replaces the superposition operator with the geometric mean. We use the union instead of the more complex ruleset of Feng et al. to decide which overlapping node parents to retain. The more straightforward allows for combining more than two BNs at a time, lowering the number of communication rounds. It also reduces the attack surface area due to fewer checks on party data. For fusing probabilities, the geometric mean enables a multiplication-based secret sharing scheme in CABN. Reconstruction happens automatically during distributed inference when merging local results from parties. The mean also outperforms the superposition in our centralized tests.

Step 1: Private Common Node Identification. CABN starts with party pairs identifying their common nodes like in the central case, albeit privately. We use a private set intersection (PSI) protocol [12] to attain confidentiality (ll. 1-6). Only parties that have updates need to recalculate their intersection with the others. Outside the private intersection context, node and state names are communicated obfuscated to prevent information leakage about which parties model which nodes. Parties can choose any unique representation for non-overlapping nodes, but involved parties agree on an obfuscated representation for overlapping ones. As outlined in Appendix C, our implementation relies on a commutative elliptic curve ElGamal cypher.

Step 2: Local Alignment. After parties know which local nodes overlap with which peers, they start solving overlaps by exchanging structure and weight information about their local CPDs and independently updating local representations accordingly (ll. 7-14). First, the obfuscated union of CPD nodes and states for overlapping CPDs is determined (ll. 8-9). From it, an identity CPD containing all the parents across parties gets created for the union (l. 10). An identity CPD (or factor) has all entries equal to 1, so its product with another replicates the later's columns over their joint state space. The initial CPD gets replaced by the product with the identity in each party, giving all overlap CPDs the same shape (l. 13), as seen in Figure 2a. Parties have a public weight representing confidence in their BN, which in the default unweighted case is 1. They compute the sum of their weights (l. 11) and individually raise the entries of their CPD to the ratio between their weight and the sum, computing the partial geometric mean (l. 14). By the exponent product rule $(XY)^k = X^k Y^k$, the CPDs' product already yields the unnormalized CPD of the central combination.

Model Weighting. As previously mentioned, CABN allows weighting CPDs through the geometric mean, unlike previous BN works that cover parameter fusion. A natural integration of unequal weighting of inputs is another advantage of using a geometric mean instead of Feng et al.'s superposition. We implement weights at the model level as 0-1 values, encoding the human expert's confi-

dence in the network or data availability for algorithmic learning. Nevertheless, weighting can be applied at the CPD level.

Step 3: Secure Normalization. Homomorphic encryption (HE) [7] allows to privately compute column normalization values (ll. 15-19). One party is elected to generate the encryption key pair and another to perform the encrypted computation. All parties receive the public key and send their encrypted columns to the party that calculates normalization value ciphers (l. 16). The private key party decrypts the values and shares them with the rest (l. 17). A column normalization value is the sum of entries obtained by multiplying matching party columns element-wise. Letting P_{ij} denote party i CPD column j, we thus find $|| \odot_i P_{ij}||_1$. Then, the K overlapping parties individually divide each column by the K-th root of the appropriate normalization value (l. 19), so their factor product is the normalized geometric mean. Figure 2a shows an example. Our implementation relies on the CKKS [7] HE scheme for floating-point addition and multiplication. We include parametrization information in Appendix C.

Because local columns no longer sum to 1 after exponentiation, even in a two-party overlap where the variable has only two possible states, a party cannot reconstruct the other's entries by only knowing the normalization values and its own entries. Furthermore, vital for HE schemes in practice, we know that the number of consecutive multiplications needed for each column is equal to the party count, which allows configuring the scheme accordingly. Functional encryption, in which completing the desired computation also decrypts the output [5], can be a more viable choice, but existing implementations have overly stringent limits on the number of inputs and complexity of the applied functions. Using a secret sharing scheme (SSS) [9] instead of HE is also possible, but we favor decreasing the communication count over computing overhead for this step. An SSS has the advantage of requiring fewer computational resources and being more robust against collusion. However, it requires communication for each multiplication operation and, depending on the scheme, the presence of a third party. Despite the expectation that `CABN` needs to run more rarely than inference, we still favor optimizing for message count. High communication latency is likelier to be a bottleneck than processing for envisioned deployments.

The only extra role of the two elected parties, K and C, is to manage the private/public key pair and compute with the encrypted data, respectively. We elect both parties from those in the overlap, so their CPDs are also part of the calculation. They can be picked based on any criteria, like the order of joining the network. Party K forwards the public key to all others in the overlap, C excluded. Parties, K included, send their encrypted data to C, which calculates the normalization value. The result gets forwarded to K, which decrypts and shares it with all overlap parties. Apart from the normalization value, party K only ever has the chance to decrypt data it owns. The elected parties may also be outside the overlap as long as they are different entities. Appendix A discusses alternatives for if parties can collude so the current setup is no longer suitable.

Step 4: Secret Sharing. Finally, to combat party parameter leaks at inference, we secret share [23] the CPD entries of parties in each overlap through

Algorithm 2 SAVE

Input: $P = \{p_1,\ p_2,\ \dots\}$ – parties,
$\mathcal{Q} = \{X,\ Y,\ \dots\}$ – query vars, $\mathcal{E} = \{a_i,\ b_j,\ \dots\}$ – evidence var states
Output: factor over $\mathcal{Q}$

1: **for each** party $p \in P$ **do**
2: {receive $\mathcal{Q}$ & $\mathcal{E}$ from querying party}
3: $inFacts_p \leftarrow$ set of CPDs in p as factors
4: $\mathcal{Q}_p \leftarrow$ all overlap vars of p and their parents
5: $outFact_p \leftarrow VE(\mathcal{Q} \cup \mathcal{Q}_p,\ \mathcal{E},\ inFacts_p)$ {exec by p}
6: {send $outFact_p$ to querying party}
7: **return** $VE(\mathcal{Q},\ \{\},\ \bigcup_{p \in P} outFact_p)$ {exec by querying party}

a multiplication-based scheme (ll. 20-23). The scheme allows using the product operation needed for inference while exchanging a similar number of messages to HE and achieving much better computation scaling. The common shape of updated local CPDs facilitates the procedure. In the classic additive secret sharing scheme, a secret value is split into shares distributed among parties whose sum is the secret. It allows efficient and secure computation of expressions summing multiple secret values and applying other operations involving non-secret values. Parties perform the computation with their local share of each secret and all aggregate their results to reconstruct the answer. The utilized scheme functions similarly but uses multiplication as the base operation instead. Reconstruction happens implicitly during inference, with no extra overhead since other factors containing partial results from parties get incorporated into the final result via the same product operations. We detail the scheme construction in Appendix D and exemplify the share splitting within CPDs in Figure 2a.

Handling potential cycles. To avoid compatibility restrictions between combinable BNs, if solving overlaps creates a cycle, the distributed global network gets treated as an MRF, with no changes to the inference, which operates on factors regardless. Edges that form cycles in the BN are effectively incorporated into the moralized MRF and treated as undirected. Since the main target is to not share the complete combined network, the readability advantages of BNs are not a concern in the joint global model. Deciding which edge to remove from a cycle often requires unavailable information and threatens confidentiality. Alternatives like treating all nodes within a cycle as a single node [40] are coarse-grained and threaten confidentiality.

3.2 Share Aggregation Variable Elimination

SAVE[5] is the inference protocol wherein all parties run VE locally before aggregating their outputs into the final factor. Algorithm 2 describes its steps, and Figure 2b visualizes an example query. Parties execute most of the process in parallel (algo. l. 1). They first receive the obfuscated target variables $\mathcal{Q}$ and

[5] **S**hare **A**ggregation **V**ariable **E**limination

evidence $\mathcal{E}$ from the querying party (algo. l.2, S1 in fig.). Then, they prepare all their CPDs as factors (algo l. 3). They extend the queried variables set with local overlap variables and their parents (algo. l. 4). Extending the set avoids illegal marginalization operations within our SSS. Although the individual operations have no overhead via our scheme, the changed order of operations is less efficient, increasing the cost of VE. Parties each run VE over their factors and extended query set (algo. l. 5, S2 in fig.). To reduce overall communication strain, our implementation defaults to stopping VE early and returning a list of disjoint smaller factors instead of a single large one. Consequently, each party sends its result to the querying party (algo. l. 6, S3 in fig.). Finally, the querying party runs VE with the original set of query variables and received factors to get the query result factor (algo. l. 7, S4 in fig.).

Appendix D further describes the interaction between VE and secret sharing. Applying a *log* transform to values and using additive secret sharing instead would also work for products but cause even more marginalization issues. Allowing multiplications between shares and plaintext would require taking the *log* of all values, not just shares. Consequently, plaintext marginalization becomes impossible since addition has no correspondence under *log*. Fixed-point arithmetic keeps the scheme's theoretical guarantees by mapping fractional CPD values to integers. Within it, rounding avoids overflows from multiplication chains.

Queryable Nodes. To maintain confidentiality, parties can only specify modeled variables in inference by default, even if the result still reflects the effect of prior knowledge about others, so we propose mechanisms for expanding the set of possible queries. The first involves all parties that own a node agreeing to expose its unobfuscated name and states with select others to use as a target or evidence. Doing so only requires revealing a node's existence, not its place in the network(s). The other mechanism implicitly enhances evidence with the help of some key shared between parties (e.g., timestamp, product batch identifier). If a query request also includes a value for the shared key, parties incorporate any observations for the key's value as evidence during their local inference step. Parties do not have to disclose the value of the observed data, but the query output is the same as if it had been part of the initial evidence.

Communication Properties. The number of messages exchanged within an inference request is of magnitude $O(N)$ (where N is the number of parties), but the size of the messages varies. Regarding count, the requester sends out $N-1$ messages and receives the same amount of replies adding up to $2N-2$. The size of the messages, particularly replies, varies greatly depending on the number and complexity of the responder's overlaps and the query itself.

4 Performance Evaluation

Our main results show `CCBNet` achieving the predictive ability of centralized solutions in the confidential, collaborative setting and examine its ability to maintain reasonable computation and communication costs. Appendix E contains further experimental details and additional results.

Class	Name	#Nodes	#Edges	#Params
Small (< 20 Nodes)	ASIA	8	8	18
Medium (20-49 Nodes)	CHILD	20	25	230
	ALARM	37	46	509
	INSURANCE	27	52	1008
Large (50-99 Nodes)	WIN95PTS	76	112	574
Very Large (101-999 Nodes)	ANDES	223	338	1157
	PIGS	441	592	5618
	LINK	724	1125	14211
Massive (≥ 1000 Nodes)	MUNIN2	1003	1244	69431

Table 1: Evaluation datasets with node/edge/parameter counts

4.1 Setup

Our experiments evaluate average predictive performance, computation overhead, and communication cost of inference in single-machine simulations. We measure prediction quality based on the Brier Score ($= \frac{1}{N}\sum_{t=1}^{N}\sum_{i=1}^{R}(f_{ti}-o_{ti})^2$, where N is the number of queries, R is the number of target variables state combinations, while f and o are predicted and reference probabilities). Note that, different from the original multi-category definition, we have probability values for all reference entries o instead of a binary value differentiating an observed (1) state from all others (0). We report the total processing time ratio between examined methods and the ground truth network for computation overhead. For communication we consider the count of factor values exchanged per query. We do not evaluate CABN time or communication as we expect it to be amortizable.

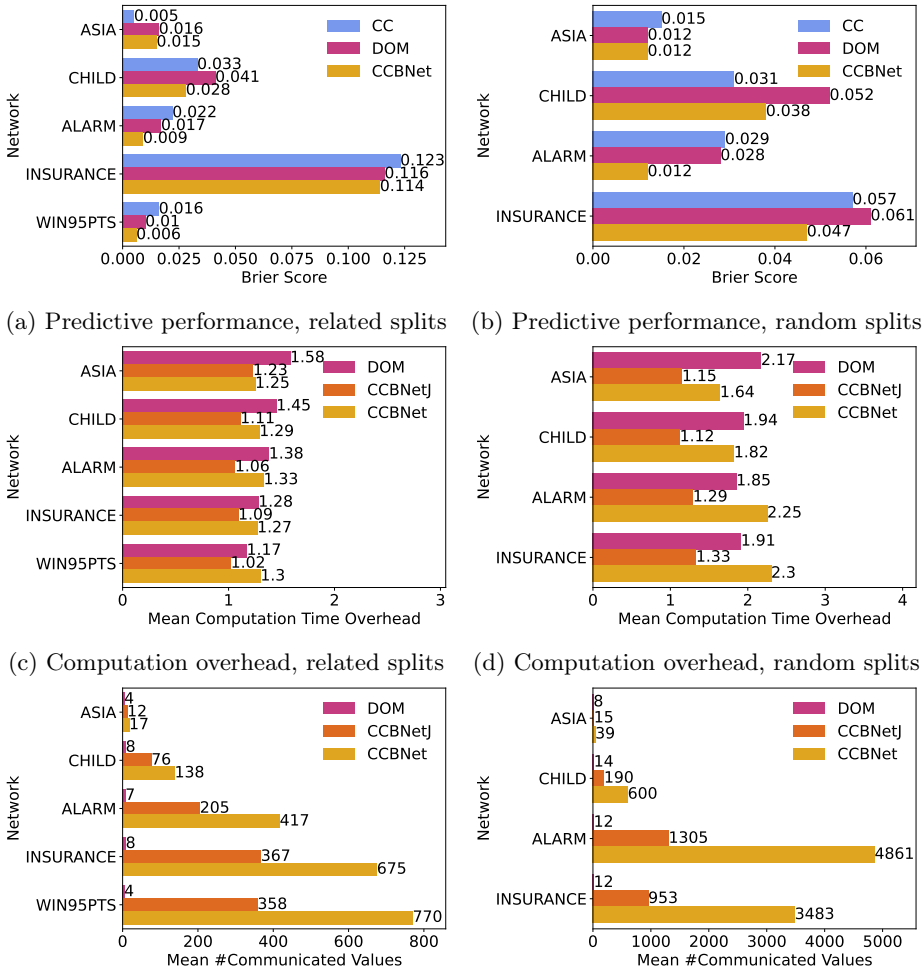
We test on public networks[6] (Table 1), and consider two variable splitting methods alongside multiple overlap variable ratios. **Related splits** assign to parties variables connected in the ground truth network. **Random splits** ensure parties have equal variable counts and share the same overlaps. Test sets contain 2000 queries, each with one random overlap variable as the target and 60% of the others fixed as evidence. Related splits attempt to simulate a realistic deployment in which the experts within clients have an incomplete but close-to-the-ground truth view of the interactions between their modeled variables (which also differ in number). Random splits aim for a worst-case scenario where the variables within clients (of roughly equal number) are not subgraphs of the complete network, generally giving more densely connected local networks and overlaps. Appendix C describes the split procedures more formally.

4.2 Baselines

As the closest prior work is strictly centralized, our baselines are:

Centralized Combination (CC) iteratively combines parties by the method of [16] and treats the network as an MRF if cycles form.

[6] https://www.bnlearn.com/bnrepository/



(a) Predictive performance, related splits (b) Predictive performance, random splits

(c) Computation overhead, related splits (d) Computation overhead, random splits

(e) Communication cost, related splits (f) Communication cost, random splits

Fig. 3: Performance of decentralized methods relative to ground truth (lower is better); 4 parties, 30% of vars in > 1 party

Centralized Union (CU), the approach confidentially mimicked by `CCBNet`, is structurally based on the union of [13], combines parameters via geometric mean, and also applies the MRF principle.

Decentralized Output Mean (DOM) is a naïve approach that takes the geometric mean for each target variable's state probabilities over independently operating contained parties, trading long-range effect modeling for greater safety.

`CCBNetJ` is a degenerate `CCBNet` variant that stores fully combined central CPDs for overlaps in singular parties, trading some safety for faster inference.

4.3 CCBNet Performance Overview

Below we summarize the Brier score, computation time, and communication cost of CCBNet under two splitting methods, different overlapping ratios (10%, 30%, and 50%), and party numbers (2, 4, and 8). Due to the space limitations, we present the full results in the appendix and highlight in Figure 3 the performance trend via the representative case of four parties with a 30% overlap ratio.

Predictive Performance. Regardless of split type, CCBNet predictions often outperform or match the classic CC and always beat the naïve DOM. Figure 3a gives results for related splits, CCBNet only scores worse than CC on the smallest network (0.015 versus 0.005) and gains a significant advantage over the larger ones (0.022 versus 0.009, 0.123 versus 0.114, and 0.006 versus 0.016). CCBNet has an advantage over DOM in all scenarios. Examining the random splits in Figure 3b, CCBNet still outscores CC in all but one of the smaller networks (0.031 vs 0.038) and maintains its lead over DOM in all but the smallest network, where the two tie. Since CCBNet yields the same predictive ability as its centralized counterpart CU, the differentiating points with CC are the structure combination policy and parameter combination operator. Over both splits, the pairing of the union selection and geometric mean aggregation in CCBNet fares better in most cases than the nondeterministic selection and superposition aggregation of CC, apart from some instances within small networks. Finally, the appendix results confirm the expectation that adding more parties tends to decrease performance while increasing overlaps has the opposite effect.

Computation Overhead. Regarding computation cost relative to centralized inference on the original network, average slowdowns are 1.63x for DOM, 1.15x for CCBNetJ, and 1.6x for CCBNet. Communication latency is unaccounted for as it can vary greatly based on the deployment. Still, we overestimate wall time by summing computation time across parties, as much processing would happen concurrently in reality. In the related splits from Figure 3c, across the board, CCBNetJ is the fastest, but others follow closely. CCBNet is the second best in all except the largest networks, where it fares worse than DOM (1.17x vs 1.3x slower). The networks also examined for random splits in Figure 3d show a similar trend, with somewhat higher general overhead, especially for CCBNet. Thus, as expected, CCBNetJ is faster than CCBNet, but both perform reasonably in most scenarios, even if the latter suffers more as overlaps increase in complexity. The DOM implementation does has a higher base overhead, but the gap to the other algorithms is often relatively contained. Appendix tests certify that adding parties improves speed while increasing overlaps decreases it. As for absolute total computation time, queries are slowest on the largest related splits dataset, as DOM averages 2.9 ms/query, CCBNetJ 2.5 ms, and CCBNet 3.2 ms.

Communication Cost. DOM averages merely 9 CPD values communicated per query, while CCBNetJ and CCBNet need orders of magnitude more at 387 and 1222, respectively. Since the number of communicated values depends on which party initiates a query, the reported figures include communication within the querying party to eliminate variability but overestimate reality. The number of messages to complete a query is the same for all methods. Furthermore, the

	#Parties	2				4			
	Vars in >1 Party	10%		30%		10%		30%	
	Method	UW	W	UW	W	UW	W	UW	W
Network	ASIA	0.028	**0.018**	0.029	**0.009**	0.036	**0.020**	0.031	**0.011**
	CHILD	**0.067**	**0.067**	0.088	**0.047**	0.084	**0.067**	0.097	**0.048**
	ALARM	0.063	**0.058**	0.061	**0.050**	0.103	**0.092**	0.061	**0.057**
	INSURANCE	**0.079**	0.104	**0.051**	**0.051**	0.112	**0.088**	0.135	**0.098**

Table 2: Unweighted **(UW)** & Weighted **(W)** Brier scores for `CCBNet` relative to original network – random splits, imbalanced learning data (lower is better)

raw data transmitted in bytes remain in the low megabyte range for hundreds of thousands of values before compression. Figure 3e shows the mentioned discrepancy over related splits for all but the smallest network, in which the three methods are comparable. DOM merges complete party outputs and cannot propagate evidence between parties. Thus, it does not increase communication with the number of overlaps, and parties that do not contain any target variables send empty replies. The situation for random splits, illustrated in Figure 3f, is very similar, although the disadvantage of `CCBNet` over `CCBNetJ` widens considerably. Communication costs and the overlap ratio (except for DOM) increase with the number of parties in all methods (cf. Appendix E).

4.4 Party Weighting

Table 2 shows the weighted version of the proposed method having better predictive performance than the unweighted one in almost all scenarios with random splits. In weighting tests, we reduce the overall amount of data used for learning local BNs to ensure more variance and alternatingly assign parties a smaller or larger fraction of training data. Over the one scenario where the unweighted version performs better (0.079 versus 0.104), parties with lower data get overly punished for their perceived imprecision. Since each CPD has a single weight, all parents of a node within the party are still treated uniformly according to that value, even if there is a mismatch between the weight and the actual quality of the estimates. Similarly, if a party with lower overall confidence is the only one to model a highly influential parent, its importance is underrepresented in the final result. Nevertheless, despite these phenomena, which can adversely affect performance, weighting has a positive overall impact in tested scenarios.

4.5 Large Networks & Many Parties

Lastly, in Table 3, our tests for challenging use cases with large networks (223-1003 variables), many parties (16-28), and related variable splits, but lower overlaps reconfirm prediction/communication trends, yet computation improves over original networks. In larger networks, `CCBNet`'s predictions outperform CC, and communication size increases with parties and network size, averaging 71k values

Networks/ Parties	Brier Score			Avg Comp Time Overhead			Avg #Comm Values		
	CC	DOM	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
ANDES/16	0.040	0.039	**0.035**	0.58	0.62	0.68	4	317	445
PIGS/32	0.103	0.092	**0.076**	0.39	0.51	1.34	7	4202	37838
LINK/64	0.144	**0.125**	0.126	0.25	0.35	0.39	6	2402	4549
MUNIN2/128	0.016	**0.015**	**0.015**	0.20	0.41	0.67	11	88581	242788

Table 3: Metrics for large networks & many parties – related splits, 10% of vars in >1 party (lower is better)

per request. Computation overhead is always <1, for a mean speedup of 21%. The hardness of inference makes approximating a big network by splitting it into chunks faster, even before considering parallel party solving.

5 Attacks on CCBNet

Carelessly combining related BNs into a single global model has a high risk of leaking confidential information to all parties that can access it. As illustrated in the middle of Figure 1: (i) at a purely structural level the centralized combination can contain a large amount of the local party information; and (ii) at a parameter level, probability functions for any non-overlap nodes remain unmodified. Since BNs are human-readable, inspection can compromise sensitive information even before any inference. We review two attacks that respectively reconstruct CPDs during CABN and SAVE, along with their implications in CCBNet and CCBNetJ. The attacks do not bypass the obfuscation of unowned variable names and states but still expose potentially sensitive information via the recovered probability values. We execute these attacks as visualized in Figure 4 on a corner case of **two-party ASIA and CHILD network instances**.

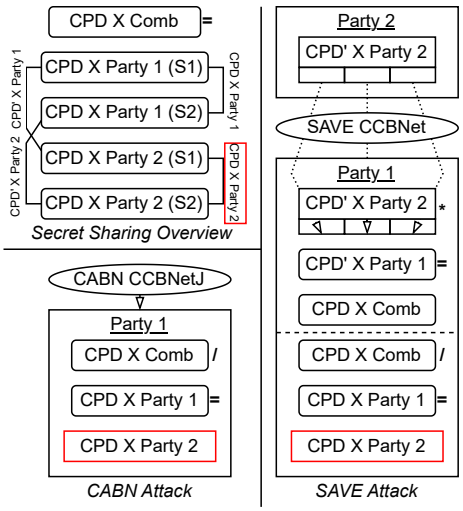


Fig. 4: CABN & SAVE attacks visualization

The **CABN attack** allows a party to reconstruct a peer's CPD for an overlap variable without inference, assuming no other parties are involved in the overlap, the attacker holds the combined CPD, and the protocol is CCBNetJ. The attacker first runs CABN to compute the combined CPD. Then, it removes its contribution from the geometric product that yielded the CPD, marginalizes any parents that should not be present in the victim's version, and normalizes to get the

final result. With any secure computation scheme, if one input parties knows the output and the applied operations are reversible, it can find the other input. With three or more parties, the attacker can only reconstruct an aggregation of the other involved CPDs. Even when the attack is possible, name obfuscation still hides the real-world meaning of the variables. Should any party outside the overlap exist, in the assumed no collusion setting, having it compute and store the combined CPD instead gives a minimum-change fix. `CCBNet` is not vulnerable, as it does not join shares before applying inference operations.

The **`SAVE` attack** reconstructs a peer's CPD for an overlap variable via inference assuming that the overlap contains no other parties but also applies to `CCBNet`. As one of the two parties in the overlap, the attacker begins by querying for the overlap variable as the only target, specifying a state for each of its parents in the evidence. With all parents fixed, no other variables in the victim's network can affect the transmitted result. Thus, the attacker receives one row from the victim's share of the combined target CPD. The attacker repeats the procedure for all other configurations of parent states, building up the victim's complete CPD share. It then obtains the full combined CPD as the product of its share and the one recovered from the other party. Finally, like in the previous attack, it removes the contribution of its whole local CPD from the combination, marginalizes any parents not in the victim, and normalizes to find the result. Also similar to the previous attack, obfuscation limits the damage that can be done, while with the involvement of more than two parties, the attacker is able to compromise their shares, but not the contribution of each, before sharing the secrets. Thus, secret sharing avoids the attack for overlaps with three or more parties. As the attack requires a series of specific queries, redundant in most real applications, a simple defense has parties limit the number of requests serviced that target some node and assign all parents in the evidence.

6 Conclusion

We propose `CCBNet` to address the issue of collaborative analysis for BNs in confidential (manufacturing) settings. The framework allows distributed analysis spanning multiple models without revealing their contents. It has no model compatibility restrictions and allows unequally weighting parties. Results show `CCBNet` outperforming/matching distributed/centralized baselines while maintaining reasonable computation time that decreases to 23% of centralized formulations in large networks with many parties but produces much larger messages the distributed baseline, averaging 71k communicated values per request. Altogether, `CCBNet` offers centralized-like predictive performance in a distributed setting and ensures a base for protecting parties' private information.

Acknowledgments

This research was partly funded by the NWO Perspectief project, DepMAT, and the SNSF project, Priv-GSyn 200021E_229204.

References

1. Abyaneh, A., Scherrer, N., Schwab, P., Bauer, S., Schölkopf, B., Mehrjou, A.: Fed-cd: Federated causal discovery from interventional and observational data (2022). https://doi.org/10.48550/ARXIV.2211.03846, https://arxiv.org/abs/2211.03846
2. Alrajeh, D., Chockler, H., Halpern, J.Y.: Combining experts' causal judgments. Artificial Intelligence **288**, 103355 (2020). https://doi.org/https://doi.org/10.1016/j.artint.2020.103355, https://www.sciencedirect.com/science/article/pii/S0004370220301065
3. Battiston, I., Felius, L., Ansmink, S., Kuiper, L., Boncz, P.A.: Duckdb-sgx2: The good, the bad and the ugly within confidential analytical query processing. In: Binnig, C., Tatbul, N. (eds.) Proceedings of the 20th International Workshop on Data Management on New Hardware, DaMoN 2024, Santiago, Chile, 10 June 2024. pp. 14:1–14:5. ACM (2024). https://doi.org/10.1145/3662010.3663447, https://doi.org/10.1145/3662010.3663447
4. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91. pp. 420–432. Springer Berlin Heidelberg, Berlin, Heidelberg (1992)
5. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) Theory of Cryptography. pp. 253–273. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
6. Campbell, T., How, J.P.: Approximate decentralized bayesian inference (2014)
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 409–437. Springer International Publishing, Cham (2017)
8. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Physical Review E **70**(6) (dec 2004). https://doi.org/10.1103/physreve.70.066111, https://doi.org/10.1103%2Fphysreve.70.066111
9. Cramer, R., Damgård, I.B., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press (2015). https://doi.org/10.1017/CBO9781107337756
10. van Daalen, F., Ippel, L., Dekker, A., Bermejo, I.: Vertibayes: Learning bayesian network parameters from vertically partitioned data with missing values (2022). https://doi.org/10.48550/ARXIV.2210.17228, https://arxiv.org/abs/2210.17228
11. Daly, R., Shen, Q., Aitken, S.: Learning bayesian networks: approaches and issues. The knowledge engineering review **26**(2), 99–157 (2011)
12. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Sion, R. (ed.) Financial Cryptography and Data Security. pp. 143–159. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
13. Del Sagrado, J., Moral, S.: Qualitative combination of bayesian networks. International Journal of Intelligent Systems **18**(2), 237–249 (2003). https://doi.org/https://doi.org/10.1002/int.10086, https://onlinelibrary.wiley.com/doi/abs/10.1002/int.10086
14. Escalera, D.M., Agudo, I., López, J.: Private set intersection: A systematic literature review. Comput. Sci. Rev. **49**, 100567 (2023). https://doi.org/10.1016/J.COSREV.2023.100567, https://doi.org/10.1016/j.cosrev.2023.100567
15. Fei, S., Yan, Z., Ding, W., Xie, H.: Security vulnerabilities of SGX and countermeasures: A survey. ACM Comput. Surv. **54**(6), 126:1–126:36 (2022). https://doi.org/10.1145/3456631, https://doi.org/10.1145/3456631

16. Feng, G., Zhang, J.D., Shaoyi Liao, S.: A novel method for combining bayesian networks, theoretical analysis, and its applications. Pattern Recognition **47**(5), 2057–2069 (2014). https://doi.org/https://doi.org/10.1016/j.patcog.2013.12.005, https://www.sciencedirect.com/science/article/pii/S0031320313005232
17. Gao, E., Chen, J., Shen, L., Liu, T., Gong, M., Bondell, H.: Feddag: Federated dag structure learning (2021). https://doi.org/10.48550/ARXIV.2112.03555, https://arxiv.org/abs/2112.03555
18. Gholami, B., Yoon, S., Pavlovic, V.: Decentralized approximate bayesian inference for distributed sensor network. Proceedings of the AAAI Conference on Artificial Intelligence **30**(1) (Feb 2016). https://doi.org/10.1609/aaai.v30i1.10201, https://ojs.aaai.org/index.php/AAAI/article/view/10201
19. Goldreich, O.: Foundations of cryptography – a primer. Foundations and Trends® in Theoretical Computer Science **1**(1), 1–116 (2005). https://doi.org/10.1561/0400000001, http://dx.doi.org/10.1561/0400000001
20. Hosseini-Khayat, S.: Using commutative encryption to share a secret. IACR Cryptol. ePrint Arch. p. 356 (2008), http://eprint.iacr.org/2008/356
21. Huang, J., Yu, K., Guo, X., Cao, F., Liang, J.: Towards privacy-aware causal structure learning in federated setting (2022). https://doi.org/10.48550/ARXIV.2211.06919, https://arxiv.org/abs/2211.06919
22. Kales, D.: Secret sharing, https://www.iaik.tugraz.at/wp-content/uploads/teaching/mfc/secret_sharing.pdf
23. Kilbertus, N., Gascon, A., Kusner, M., Veale, M., Gummadi, K., Weller, A.: Blind justice: Fairness with encrypted sensitive attributes. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 2630–2639. PMLR (10–15 Jul 2018), https://proceedings.mlr.press/v80/kilbertus18a.html
24. Kim, H.C., Ghahramani, Z.: Bayesian classifier combination. In: Lawrence, N.D., Girolami, M. (eds.) Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 22, pp. 619–627. PMLR, La Palma, Canary Islands (2012), https://proceedings.mlr.press/v22/kim12.html
25. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)
26. Li, S.Z.: Markov random field modeling in image analysis. Springer Science & Business Media (2009)
27. Ma, J., Naas, S., Sigg, S., Lyu, X.: Privacy-preserving federated learning based on multi-key homomorphic encryption. Int. J. Intell. Syst. **37**(9), 5880–5901 (2022). https://doi.org/10.1002/INT.22818, https://doi.org/10.1002/int.22818
28. Ménétrey, J., Pasin, M., Felber, P., Schiavoni, V.: Twine: An embedded trusted runtime for webassembly. In: 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021. pp. 205–216. IEEE (2021). https://doi.org/10.1109/ICDE51399.2021.00025, https://doi.org/10.1109/ICDE51399.2021.00025
29. Nannapaneni, S., Mahadevan, S., Rachuri, S.: Performance evaluation of a manufacturing process under uncertainty using bayesian networks. Journal of Cleaner Production **113**, 947–959 (2016). https://doi.org/https://doi.org/10.1016/j.jclepro.2015.12.003, https://www.sciencedirect.com/science/article/pii/S0959652615018144
30. Ng, I., Zhang, K.: Towards federated bayesian network structure learning with continuous optimization (2021). https://doi.org/10.48550/ARXIV.2110.09356, https://arxiv.org/abs/2110.09356

31. Pavlin, G., de Oude, P., Maris, M., Nunnink, J., Hood, T.: A multi-agent systems approach to distributed bayesian information fusion. Information Fusion **11**(3), 267–282 (2010). https://doi.org/https://doi.org/10.1016/j.inffus.2009.09.007, https://www.sciencedirect.com/science/article/pii/S1566253509000864, agent-Based Information Fusion
32. Pearl, J.: Causality. Cambridge University Press, 2 edn. (2009). https://doi.org/10.1017/CBO9780511803161
33. Russell, S.J.: Artificial intelligence a modern approach. Pearson Education, Inc. (2010)
34. Sardar, M.U., Fetzer, C.: Confidential computing and related technologies: a critical review. Cybersecur. **6**(1), 10 (2023). https://doi.org/10.1186/S42400-023-00144-1, https://doi.org/10.1186/s42400-023-00144-1
35. Scutari, M., Denis, J.B.: Bayesian networks: with examples in R. CRC press (2021)
36. Stephenson, T.A.: An introduction to bayesian network theory and usage. Tech. rep., Idiap (2000)
37. Tedesco, R., Dolog, P., Nejdl, W., Allert, H.: Distributed bayesian networks for user modeling. In: Reeves, T., Yamashita, S. (eds.) Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2006. pp. 292–299. Association for the Advancement of Computing in Education (AACE), Honolulu, Hawaii, USA (10 2006), https://www.learntechlib.org/p/23699
38. Vaswani, K., Volos, S., Fournet, C., Diaz, A.N., Gordon, K., Vembu, B., Webster, S., Chisnall, D., Kulkarni, S., Cunningham, G., Osborne, R., Wilkinson, D.: Confidential computing within an AI accelerator. In: Lawall, J., Williams, D. (eds.) Proceedings of the 2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023. pp. 501–518. USENIX Association (2023), https://www.usenix.org/conference/atc23/presentation/vaswani
39. Yang, W.T.: An Integrated Physics-Informed Process Control Framework and Its Applications to Semiconductor Manufacturing. Theses, Université de Lyon (Jan 2020), https://theses.hal.science/tel-03461289
40. Ypma, A., Koopman, A.C.M., Middlebrooks, S.A.: Methods & apparatus for obtaining diagnostic information, methods & apparatus for controlling an industrial process (Feb 2018)

A Usability Discussion

We provide further context regarding the framework's flexibility and appropriateness for real-world deployments:

Framework Flexibility. As outlined through the rest of the work, the framework naturally supports more variations than the presented `CCBNet` and `CCBNetJ`, allowing for a choice between increased confidentiality measures or other performance criteria. One choice entails replacing the HE with an SSS in `CABN` to swap computation by communication. Another is disallowing specific two-party overlap queries to avoid compromising network parameters.

Furthermore, running the inner loop VE calls in `SAVE` to completion would increase security at the cost of more overhead. Thus, instead of returning a set of factors, each party would directly return their product. Since the factors often have mostly disjoint variables, the product contains more numerical entries to

transmit than when they are separate. Nevertheless, it also makes it harder for the receiver to gain insight into the sender's independent variables. Finally, while not tested within our initial implementation, a dynamic approach could provide an even better safeguard and overhead balance. It would choose between the product and keeping the factors separate based on an inexpensive preliminary check for which option requires the fewest entries.

Framework Applicability. Although our motivating use case comes from semiconductor manufacturing, we devise `CCBNet` to be as generally applicable as possible, even outside the (semiconductor) manufacturing industry. Hence, we also base our evaluation on public data sets. Future improvements to further aid such goals could be confidentially harnessing available observed data samples within parties when updating overlap variable representation, allowing additional types of workloads on the representation (e.g., approximate inference), or natively supporting continuous variables types.

Adversarial Setting. Although the considered semi-honest (passive), no-collusion adversarial setting restricts the scenarios in which the protocol is adequate, we deem it helpful in many manufacturing contexts and a stepping stone for more robust formulations. Unlike classic peer-to-peer systems with anonymous participants, those involved in such industrial scenarios are business partners known to each other. They generally have little incentive to sabotage the system's functioning, as the disruption would also negatively impact them, lessening the need for a solution against malicious (active) adversaries. Furthermore, we currently focus on scenarios where each entity (e.g., manufacturer) joins the collaboration network as one party, meaning collusion would require coordination between multiple entities with individual interests.

Moving to the unrestricted semi-honest scenario only requires updating `CABN`'s secure normalization, while the malicious case involves more significant changes, also to `SAVE`, likely even with additional auxiliary assumptions. Should its feasibility for the normalization workload be verified, an alternative like multi-key HE [27], where all parties must participate in decryption or another secret sharing formulation, would remove the collusion assumption. The current setup fails when the elected party colludes with the one tasked with decryption. The secret sharing step tied into inference already tolerates all but two parties colluding. With malicious adversaries, maintaining confidentiality should require minimal changes, but ensuring qualitative query outputs (e.g., by ignoring malicious inputs) would be challenging in the general case. Parties would have to detect malicious behavior when solving overlaps or performing inference and agree on the culprit(s) without access to ground truth or even the original networks of suspected parties. Honest parties with higher deviations (in a positive or negative direction) from peers' behavior would likely risk getting identified as malicious.

B Confidential Computing Comparison

Confidential computing denotes the process of protecting sensitive data during use, generally intended to be via a hardware-based trusted execution environ-

ment (TEE) called a secure enclave [34]. Variants are available from all prominent CPU/SoC vendors with varying capabilities. They all aim to isolate data in a secure part of memory with only authorized code accessing it. Albeit less common and often more limited, solutions for accelerators, like GPUs, also exist [38].

Although TEEs offer a good way of securing workloads while minimizing conceptual changes in their operation, we consider the compatibility and formality benefits of a solution built around cryptographic multi-party computation primitives essential in business settings. Although there has been considerable progress on the software that can run on enclaves [28], the heterogeneity in the features supported for different hardware (e.g., amount of accessible memory [3]) can be problematic for a multi-party system, where each may have different hardware. The issue becomes even more prominent when integrating accelerators to speed up inference. Despite often practically adequate security properties, TEEs do not offer the same formal guarantees as theory-based approaches. Many have suffered from different vulnerabilities over the years [15]. Thus, in critical (business) applications, formulations that allow deriving stricter guarantees are preferable.

Nevertheless, in scenarios where the above concerns are not prohibitive, merging the proposed solution with TEEs can lead to a better solution than using either standalone. Instead of using expensive HE during normalization in `CABN`, the elected party could receive data from others directly in the enclave, decrypt it, process the result, and then move it outside for sharing. Even though TEEs still have various overheads, which include performing system calls or working with higher memory amounts, we expect that there would be considerable performance gains compared to the HE case. In `SAVE`, however, using secret-sharing and distributing computation among parties could remain favorable to performing all inference in the enclave of one party, despite the more efficient order of operation that allows. Finally, the secure attestation features in TEEs could be helpful when moving beyond semi-honest adversaries.

C Implementation Notes

We implement the codebase in Python 3.10, using pgmpy 0.1.24[7] as the backbone for BNs in our framework, openmined-psi[8] 2.0.2 for private set intersection, and tenseal[9] 0.3.14 for homomorphic encryption. In Algorithm 3, we describe the procedure for VE, as called upon within Algorithm 2.

Private Set Intersection. We do private set intersection in openmined-psi via commutative encryption [14,20]. Specifically, it uses a commutative elliptic curve ElGamal cipher, based on the elliptic curve Diffie-Hellman, as implemented in Google's private-join-and-compute[10] library. The chosen elliptic curve is secp256r1, also known as P-256 or prime256v1. The inputs are the n and m strings with the unaltered variable names within the two parties involved in

[7] https://github.com/pgmpy/pgmpy
[8] https://github.com/OpenMined/PSI
[9] https://github.com/OpenMined/TenSEAL
[10] https://github.com/google/private-join-and-compute

an intersection. We have the algorithm reveal the contents of the intersection, not only the size of the set, so the outputs are strings with the shared variable names. We communicate raw encrypted messages instead of using Bloom Filters or Golomb Compressed sets as containers. Doing so removes any small chances of false positives but has higher communication costs, with complexity $O(\max(n, m))$. Computation complexity is $O(nlog(n) + max(n, m))$. We must perform such intersection for all pairs of parties.

Homomorphic Encryption. We use the CKKS scheme in tenseal, which wraps the Microsoft SEAL[11] C++ library. The scheme satisfies our requirement for encrypted floating-point addition and multiplication. As noted in the Microsoft APSI documentation[12], another library built on top of SEAL, the latter does not allow running with insecure parameters. Nevertheless, they need careful setting for the computation to execute successfully and sufficiently accurately. As we know the required multiplicative depth in each case, we use CKKS as a leveled HE scheme instead of full HE for increased efficiency and precision. We fix bit precision after the decimal point at 40 and dynamically compute the three necessary parameters. The first parameter is the list of coefficient modulus bit sizes. The length of inner elements gives the maximum number of successive multiplications allowed, which, in our case, corresponds to the number of parties involved in normalizing an overlapping variable. The maximum possible normalization value is the number of target variable states, and its bit length is the precision required before the decimal point. All inner elements share a value, while the outer ones share another, each calculated based on the bit precision before and after the dot. The second parameter is the polynomial modulus degree. Its possible values are lower bound by the sum of elements in the prior parameter. The aforementioned APSI also summarizes the specific thresholds. Since the performance of the HE is inversely proportional to the selected value, we choose the lowest possible valid one. The third parameter, global scale, is simply two to the inner element value within the coefficient modulus list. The inputs are matching columns from the extended CPD of all parties in the overlap. The number of elements in a column is the number of states of the target variable. The output is the normalization value for the given columns. Solving an overlap requires computing a normalization value for each column in the extended CPD. Multiplying the number of states for all parents gives the number of columns. Furthermore, the number of parties in the overlap linearly increases computation/communication overhead.

D Multiplication-based Secret Sharing

Definition 1 (Group). *A group [22] is a set* $G(\neq \emptyset)$ *and operation* $\bullet : G \times G \longrightarrow G$ *such that:*

1. *Asociativity:* $a \bullet (b \bullet c) = (a \bullet b) \bullet c, \forall a, b, c \in G$

[11] https://github.com/microsoft/SEAL
[12] https://github.com/microsoft/APSI/blob/main/README.md

Algorithm 3 VE (Variable Elimination)

Input: $\mathcal{Q} = \{X,\ Y,\ \dots\}$ – query vars, $\mathcal{E} = \{a_i,\ b_j,\ \dots\}$ – evidence var states, $\mathcal{F} = \{f_1,\ f_2, \dots,\ f_k\}$ – factors
Output: factor over $\mathcal{Q}$

$leftoverFacts \leftarrow \varnothing$
$leftoverVars \leftarrow$ set of all vars from factors in $\mathcal{F}$
$leftoverVars \leftarrow remainingVars \setminus \mathcal{Q}$
for each factor $f \in \mathcal{F}$ **do**
  $fReduced \leftarrow$ factor f with all var states from $\mathcal{E}$ reduced
  $leftoverFacts \leftarrow leftoverFacts \cup fReduced$
while $|leftoverVars| > 0$ **do**
  $v \leftarrow$ next var to remove from $leftoverVars$ {based on criteria like min weight}
  $leftoverVars \leftarrow leftoverVars \setminus v$
  $factProd \leftarrow$ empty factor
  $auxFacts \leftarrow \varnothing$
  for each fact $f \in leftoverFacts$ **do**
    if var v in fact f **then**
      $factProd \leftarrow factProd \cdot fact$
    else
      $auxFacts \leftarrow auxFacts \cup f$
  $factProd \leftarrow$ marginalize v from $factProd$
  $leftoverFacts \leftarrow auxFacts \cup \{factProd\}$
return $\prod_{f \in leftoverFacts} f$

2. *Neutral element:* $\exists!\, e \in G$ *such that* $e \bullet a = a \bullet e = a$, $\forall a \in G$
3. *Inverse element:* $a \bullet a' = a' \bullet a = e$, $\forall\, a \in G$, $\exists!\, a'$, *where* e *is the neutral element*

Definition 2 ($\mathbb{Z}_n$)**.** $\mathbb{Z}_n$ *is a group under* $\{0, 1, \dots, n-1\}$ *and addition modulo (%)* n*.*

Definition 3 ($\mathbb{Z}_p^*$)**.** $\mathbb{Z}_p^*$*, for* p *prime, is a group under* $\{1, 2, \dots, p-1\}$ *and multiplication modulo (%)* p*.*

Proposition 1. *The inverse* x' *of* $x \in \mathbb{Z}_p^*$ *is* $x^{p-2} \,\%\, p$*.*

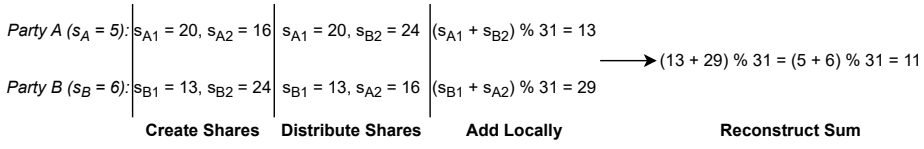


Fig. 5: Example of computing the sum of two parties' additively secret shared variables in $\mathbb{Z}_{31}$

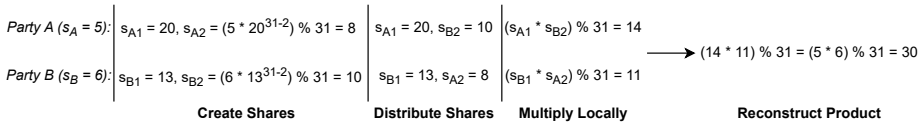


Fig. 6: Example of computing the product of two parties' multiplication-based secret shared variables in $\mathbb{Z}_{31}^*$

Secret sharing schemes are a family of methods that allow the distribution of a secret value among a group of parties by assigning each a share that does not yield any information about the secret but can, when pooled with enough others, reveal it [9]. The following details the classic additive scheme and the multiplication-based version used for `CCBNet`, relating them to each other.

Preventing information leaks about a secret with theoretical guarantees for an adversary of unbounded computation requires sampling it from a uniform distribution, which is impossible for any infinite set, like $\mathbb{Z}$ or any subset of $\mathbb{R}$. Thus, most schemes perform their computation within (derivatives of) finite integer groups (Definition 1) large enough to contain all possibly required values for computations on the secrets. Multiplication in linear schemes requires additional interaction between the parties. Usually, it involves the help of a secretly sharing an additional set of values a, b, c, called Beaver triples [4], obeying $a * b = c$, with a and b chosen arbitrarily. Protocols exist for parties to generate triples among themselves securely, but a trusted third party can simplify the procedure.

Additive [22] secret sharing is a popular scheme in literature, which is defined on $\mathbb{Z}_n$ (Definition 2), with information-theoretic security for up to $n-1$ passively corrupt parties. Figure 5 exemplifies adding two parties' secret values. Shamir's secret sharing has a configurable (t, n) threshold scheme, which produces secret shares based on a polynomial whose constant coefficient is the secret, and all others are random. It has information-theoretic security for up to $\lfloor n/2 \rfloor$ passive corrupt parties and $\lfloor n/3 \rfloor$ active ones. It uses Beaver triples to allow multiplication.

The multiplication-based scheme utilized in `CCBNet` is similar to the additive one but employs a different base operand. As it works under $\mathbb{Z}_p^*$ (Definition 3), not $\mathbb{Z}_n$, instead of agreeing on a large enough n, parties agree on a large enough prime p to instantiate the group. As in the additive case, to split a secret s into k shares, parties get shares s_1 through s_{k-1} by uniformly sampling the group's set of values and fix $s_k = s \bullet (s_1 \bullet ... \bullet s_{k-1})'$. From Proposition 1, it follows that $s_k = s * (s_1 * ... * s_{k-1})^{p-2}$, where all multiplications are modulo p. Modular exponentiation can be efficiently computed even for large exponents. Reconstruction still happens by applying the group operator to all shares. Note that, although $0 \notin \mathbb{Z}_p^*$, assuming that the party holding the secret keeps one of the shares, it can set its share to 0, and sample $\mathbb{Z}_p^*$ for the remaining ones. Figure 6 gives a small example of multiplying two parties' secret values.

In the secret sharing step of CABN, parties independently split and redistribute their shares for each value in overlap CPDs. Splitting thus distributes a CPD for an overlap between as many CPDs as the number of parties involved. With the shares defined over the same variables in the same order, their product is the element-wise multiplication of their elements and restores the initial data. For all CPDs to split, each party distributes a different share to each other party in the overlap, remaining with one for itself. As such, after parties all distribute their shares, they will have, for each shared CPD, one share of their own and one share from each other party in the corresponding overlap. The product of shares is the CPD for the corresponding variable used in inference.

During SAVE, shares multiply with plaintext values and shares from other secrets before being reconstructed into the final result. The distributed inference has two main stages, both based on VE. Parties first receive the query, independently run a round of VE over local data, and relay their intermediate results. Then the querying party runs a final round of VE inference over the received results. Within the three operations used in VE, only marginalization requires special treatment. Propagating evidence only discards entries irrelevant to the given observations. Secret sharing is applied to each value in a CPD/factor array so no issues arise. Products on secret data involve duplicating shared entries and multiplying them by other shares or plaintext. Our scheme supports such multiplications by design. Multiplication of shares relating to the same secret only happens in the final VE call within the querying party, leading to the reconstruction of the computation parties performed jointly over the shares. Marginalization generally involves addition, which is unsupported by our SSS scheme. Thus, before reconstruction, we avoid marginalizing variables that are part of a secret (those from parties overlap CPDs). Marginalizing another variable within the same factor as variables composing a secret remains possible. The sum expression is also a product of the share, so letting V_i be a state of the variable and S the share, we have: $S * V_1 + S * V_2 + .. = S * (V_1 + V_2 + ...)$.

The scheme's base input and output secret is a single numeric value. To secret share a single value, time, space, and communication is $O(n)$, for n parties (i.e., shares). The complexity is linear because each party has to receive, store, manipulate, and retransmit its share. For the algorithm, however, we apply the scheme to each element within a whole factor at once. Furthermore, each party has its own secret for a set of variables whose factor has D elements. Thus, total time and communication complexity is $O(N^2D)$ at the factor level. Space complexity, however, is only $O(ND)$ since the shares received by a party from others are all multiplied into one factor.

E Experimental Details & Additional Results

We detail the procedures used for related and random splits of ground truth variables among parties, used throughout all experiments, in Algorithm 4 and Algorithm 5, respectively. We provide additional experimental results with related splits. Table 4, Table 5, and Table 6 cover predictive performance Table 7,

Vars in >1 Party		10%			30%			50%	
Method		CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet
Network	ASIA	0.011	0.022	**0.006**	0.011	0.022	**0.006**	**0**	0.015	0.001
	CHILD	**0.002**	0.034	**0.002**	**0.001**	0.02	**0.001**	**0.001**	0.02	**0.001**
	ALARM	**0.001**	0.016	0.005	**0.001**	0.016	0.005	**0.001**	0.016	0.005
	INSURANCE	**0.007**	0.022	0.009	0.015	0.017	**0.011**	0.013	0.01	**0.008**
	WIN95PTS	0.015	0.007	**0.001**	0.01	0.007	**0.005**	0.01	0.007	**0.005**

Table 4: Brier score relative to original network – 2 parties, related splits (lower is better)

Vars in >1 Party		10%			50%		
Method		CC	DOM	CCBNet	CC	DOM	CCBNet
Network	ASIA	**0.005**	0.016	0.015	**0.013**	0.021	0.017
	CHILD	**0.08**	0.106	**0.08**	0.012	0.033	**0.011**
	ALARM	**0.027**	0.047	0.029	0.02	0.012	**0.011**
	INSURANCE	0.111	0.117	**0.108**	0.086	0.037	**0.032**
	WIN95PTS	0.019	0.011	**0.005**	0.01	0.007	**0.004**

Table 5: Brier score relative to original network – 4 parties, related splits (lower is better)

Vars in >1 Party		10%			30%			50%		
Method		CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet
Network	ASIA	**0.076**	**0.076**	**0.076**	**0.076**	**0.076**	**0.076**	0.055	**0.054**	**0.054**
	CHILD	0.165	0.173	**0.164**	**0.071**	0.09	0.072	0.044	0.046	**0.023**
	ALARM	0.04	0.074	**0.038**	0.029	0.041	**0.026**	0.039	0.022	**0.017**
	INSURANCE	0.154	0.166	**0.153**	0.189	0.135	**0.131**	0.091	0.085	**0.077**
	WIN95PTS	0.026	0.016	**0.011**	0.038	0.016	**0.011**	0.016	0.013	**0.006**

Table 6: Brier score relative to original network – 8 parties, related splits (lower is better)

Vars in >1 Party		10%			30%			50%		
Method		DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	1.29	0.93	1.07	1.53	1.11	1.26	1.61	1.1	1.31
	CHILD	1.16	1.01	1.07	1.39	1.02	1.19	1.4	1.03	1.2
	ALARM	1.12	0.98	1.08	1.13	0.99	1.08	1.16	0.98	1.05
	INSURANCE	1.18	0.99	1.09	1.36	1.07	1.31	1.56	1.11	1.49
	WIN95PTS	1.03	0.92	1.01	1.13	0.96	1.11	1.13	0.97	1.13

Table 7: Mean computation time overhead relative to original network – 2 parties, related splits (lower is better)

Vars in >1 Party		10%			50%		
Method		DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	1.74	1.11	1.27	1.85	1.12	1.42
	CHILD	1.2	1.0	1.11	1.72	1.15	1.54
	ALARM	1.11	0.99	1.07	1.44	1.1	1.44
	INSURANCE	1.18	1.06	1.19	1.66	1.2	1.65
	WIN95PTS	0.93	0.89	0.97	1.28	1.09	1.49

Table 8: Mean computation time overhead relative to original network – 4 parties, related splits (lower is better)

#Parties		8							
Vars in >1 Party		10%			30%			50%	
Method		DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	1.56	1.04	1.18	1.72	1.15	1.29	2.01	1.17	1.46
	CHILD	1.26	1.01	1.07	1.5	1.03	1.25	2.06	1.13	1.81
	ALARM	1.12	0.98	1.06	1.36	1.13	1.34	1.79	1.24	1.89
	INSURANCE	1.2	1.02	1.17	1.34	1.05	1.31	1.8	1.13	1.73
	WIN95PTS	0.89	0.85	0.91	1.18	1.04	1.37	1.45	1.08	1.73

Table 9: Mean computation time overhead relative to original network – 8 parties, related splits (lower is better)

Vars in >1 Party		10%			30%			50%	
Method		DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	4	16	22	4	16	22	4	16	24
	CHILD	10	64	80	7	72	109	7	72	109
	ALARM	5	65	93	5	65	93	5	65	93
	INSURANCE	8	117	173	7	254	417	7	359	633
	WIN95PTS	4	175	262	4	274	440	4	274	440

Table 10: Mean #communicated values rounded to nearest integer – 2 parties, related splits (lower is better)

Vars in >1 Party		10%			50%		
Method		DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	4	12	17	4	14	23
	CHILD	10	47	63	9	120	269
	ALARM	6	100	146	7	484	1038
	INSURANCE	9	432	807	8	482	979
	WIN95PTS	4	139	200	4	459	1000

Table 11: Mean #communicated values rounded to nearest integer – 4 parties, related splits (lower is better)

Vars in >1 Party		10%			30%			50%	
Method	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	4	11	15	4	11	15	4	12	20
	CHILD	10	43	59	8	60	95	10	189	604
	ALARM	6	136	205	6	227	437	8	768	2070
	INSURANCE	9	274	509	8	279	526	9	361	786
	WIN95PTS	4	115	158	5	344	1029	5	456	1588

Table 12: Mean #communicated values rounded to nearest integer – 8 parties, related splits (lower is better)

#Parties		2					4		
Vars in >1 Party		10%			30%		10%		
Method	CC	DOM	CCBNet	CC	DOM	CCBNet	CC	DOM	CCBNet
Network	ASIA	0.021	**0.012**	**0.012**	0.011	**0.006**	**0.006**	0.029	**0.023**	**0.023**
	CHILD	0.031	**0.021**	0.023	0.027	0.02	**0.018**	**0.029**	0.047	0.031
	ALARM	**0.009**	0.013	0.011	0.006	0.008	**0.005**	0.025	0.043	**0.022**
	INSURANCE	0.05	0.031	**0.028**	0.039	0.025	**0.024**	0.072	0.068	**0.052**

Table 13: Brier score relative to original network – random splits (lower is better)

#Parties		2					4		
Vars in >1 Party		10%			30%		10%		
Method	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	1.18	0.92	1.0	1.49	1.08	1.24	1.73	1.19	1.47
	CHILD	1.23	1.06	1.13	1.36	1.08	1.33	1.37	1.07	1.34
	ALARM	1.13	0.98	1.06	1.31	1.07	1.29	1.27	1.15	1.69
	INSURANCE	1.15	1.04	1.18	1.33	1.14	1.4	1.33	1.21	1.79

Table 14: Mean computation time overhead relative to original network – random splits (lower is better)

Metric		Average #Communicated Values								
#Parties		2					4		
Vars in >1 Party		10%			30%		10%		
Method	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet	DOM	CCBNetJ	CCBNet
Network	ASIA	4	11	15	4	11	16	8	16	38
	CHILD	6	97	146	7	153	254	12	80	202
	ALARM	5	189	304	6	423	691	11	714	2599
	INSURANCE	6	195	315	6	336	561	12	686	2464

Table 15: Mean #communicated values rounded to nearest integer – random splits (lower is better)

Table 8, and Table 9 covers computation overhead. Table 10, Table 11, Table 12 cover communication cost. In a few networks under related splits, adjacent overlap figures (e.g., 30% and 50%) have the same splits and inherently score, either because of the small number of nodes (ASIA) or because all possible overlaps for the given topology are formed (CHILD, ALARM). Furthermore, we include extra experiment data for each of the three metrics under random splits within Table 13, Table 14, and Table 15. For choosing the order of VE during inference, we use a min weight heuristic, which greedily chooses a variable such that the product of factors containing it has the smallest size.

We run each experiment once, with a fixed seed determining the randomness for sampling data instances from the reference network and splitting it into overlapping variables sets of parties. We utilize an Intel(R) Core(TM) i9-12900KF CPU @ 3.20GHz (note that inference in the single-machine simulation is single-threaded), with 120 GB of RAM, and Ubuntu 20.04.6 LTS as the OS.

Algorithm 4 Related Split

Input: dag – BN DAG structure,
n – number of splits/parties, $nOverlap$ - number of overlap variables
Output: list of node name sets representing each community

$dfsTree \leftarrow$ get a DFS traversal tree from dag
splits $\leftarrow$ greedily split $dfsTree$ into a list of n sets following Clauset et al. [8]
$shuffledEdges \leftarrow$ list of shuffled dag edges
for each node V from dag **do**
  $initSplit_V \leftarrow \varnothing$
for $splitN \in \{1, \ldots, nSplits\}$ **do**
  for each node $V \in splits[nrSplit]$ **do**
    $initSplit_V \leftarrow splitN$
$ovNodes \leftarrow \{\ \}$
$connSplits \leftarrow \{\ \}$
$extraEdges \leftarrow [\ ]$
for each edge $(nodeO,\ nodeI) \in shuffledEdges$ **do**
  if $|ovNodes| \geq nOverlap$ **then**
    `break`
  $edgeSplits \leftarrow \{initSplit_{nodeO},\ initSplit_{nodeI}\}$
  if $|edgeSplits| = 1$ **then**
    `continue`
  if $|connSplits| < n \wedge es \subseteq connSplits$ **then**
    $extraEdges \leftarrow extraEdges \cup (nodeO,\ nodeI)$
  else
    $ovNodes \leftarrow ovNodes \cup \{nodeO, nodeI\}$
    $connSplits \leftarrow connSplits \cup edgeSplits$
    $splits[initSplit_{nodeO}] \leftarrow splits[initSplit_{nodeO}] \cup \{nodeO\}$
    $splits[initSplit_{nodeI}] \leftarrow splits[initSplit_{nodeI}] \cup \{nodeI\}$
for each edge $(nodeO,\ nodeI) \in extraEdges$ **do**
  if $|ovNodes| \geq nrOverlaps$ **then**
    `break`
  $ovNodes \leftarrow ovNodes \cup \{nodeO,\ nodeI\}$
  $splits[initSplit_{nodeO}] \leftarrow splits[initSplit_{nodeO}] \cup \{nodeO\}$
  $splits[initSplit_{nodeI}] \leftarrow splits[initSplit_{nodeI}] \cup \{nodeI\}$
return $splits$

Algorithm 5 Random Split

Input: dag – BN DAG structure,
n – number of splits/parties, $nOverlap$ - number of overlap variables
Output: list of node name sets representing each community

$shuffledNodes \leftarrow$ list of shuffled dag nodes
$ovs \leftarrow$ list of $nOverlap$ nodes sampled without replacement from $shuffledNodes$
$splits \leftarrow$ split $shuffledNodes$ into a list of n sets
for each node name set $split \in splits$ **do**
  $split \leftarrow split \cup ovs$
return $splits$