Towards System-2 AI: Workloads and Characterizations of Energy-Based Models

Hanchen Yang[1], Jiayi Qian[1], Zishen Wan[1], Jingtian Dang[1], Ziwei Li[1], Yilun Du[2], and Tushar Krishna[1]

[1]*Georgia Institute of Technology, Atlanta, GA, USA*
[2]*Harvard University, Cambridge, MA, USA*
[1]{hanchen, jiayiqian, zishenwan, jdang39, zli669}@gatech.edu, tushar@ece.gatech.edu
[2]ydu@seas.harvard.edu

***Abstract*—The rapid progress of artificial intelligence (AI) has been largely driven by deep neural networks, yet their insufficient compositional reasoning abilities, limited robustness, and lack of explainability expose fundamental limitations for next-generation cognitive systems. Energy-Based Models (EBMs), adhering to System-2-style thinking, have recently shown remarkable performance across generative, compositional, and reasoning tasks, offering a pathway toward more robust and cognitively grounded AI. However, the system behavior of EBMs remains poorly understood. Their long sampling loops, strict execution data dependencies, model compositionality and heterogeneity make them highly inefficient on off-the-shelf hardware, creating significant barriers for scalable and real-time deployment.**

In this paper, we present the first comprehensive workload characterization and system study of EBMs. We first introduce a taxonomy that unifies all the diverse EBM algorithms (around twenty), then formalize their end-to-end runtime to expose key optimization opportunities, and experimentally profile representative models across CPU, GPU, and TPU platforms to understand their runtime breakdowns, memory behavior, computational operators, roofline model, algorithmic performance, and MCMC (Markov Chain Monte Carlo) sampling impacts. Our analysis reveals fundamental system bottlenecks unique to EBMs, including highly repetitive forward and backward neural operations, long and inefficient sampling trajectories, and severe hardware under-utilization caused by heterogeneous model components and operations. Furthermore, based on the experiments and analysis, we suggest optimization solutions across the algorithm, system and architecture levels, to improve EBM computing performance, efficiency, and scalability, aiming to pinpoint the challenges and future directions of EBM research.

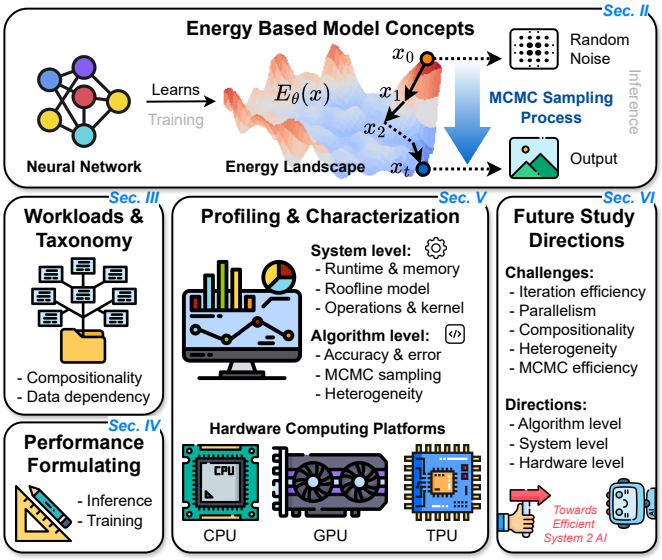


Fig. 1: Overview of Energy Based Models (EBM) concepts, workload taxonomy, performance formulating, profiling and characterizations, optimization and research opportunities in improving the performance of next-generation System-2 AI.

I. Introduction

Modern Generative AI systems are powered by enormous datasets and massive computational budgets, yet they frequently underperform on tasks that humans consider trivial, particularly tasks that diverge from the patterns seen during training [1], [2]. Humans, by contrast, do not rely solely on learned associations. Instead, human cognition flexibly alternates between two complementary modes: System 1, which reacts quickly based on pattern recognition and past experience, and System 2, which engages deliberate reasoning, step-by-step analysis, and structured problem solving when encountering unfamiliar scenarios [3]–[8].

Classical Symbolic AI frameworks have long attempted to operationalize the slower, more reflective System-2 style of reasoning through explicit search, planning, and logic-driven inference [9]–[19]. In contrast, today's mainstream generative models behave primarily as powerful System-1 engines: they excel at interpolation within the training distribution but often struggle to extrapolate or to robustly solve multi-step, compositional unseen tasks [20], [21]. This gap has motivated a search for architectures that can blend the strengths of both paradigms.

Energy-Based Models (EBMs), with diffusion models representing one notable instantiation [22]–[24], offer a promising route for embedding System-2-like computation directly into learned generative systems [23]–[40]. Rather than mapping inputs to outputs directly, an EBM trains a Neural Network (NN) to score candidate outputs with a scalar energy that reflects their satisfaction to the task and constraints. Producing a final answer for a new problem thus becomes a traversal searching for the lowest-energy on the energy landscape learned by the NN (Fig. 1). Such framework converts prediction into verification: the model does not need to know how to generate the correct solution outright, it must only evaluate how "good" or "plausible" a candidate solution is. Verification-

style functions tend to generalize more robustly to novel and out-of-distribution tasks. For example, even without being able to write a paper in a new research domain, humans can usually recognize whether one is coherent and well structured. EBMs exploit the same principle by learning to judge rather than to directly produce, thereby injecting a form of structured reasoning coupling both System 1 (judging) and System 2 (thinking) into modern generative pipelines.

EBMs provide several fundamental advantages over conventional machine learning (ML) paradigms. ❶ **Compositionality**. Because EBMs encode constraints and preferences as energy functions, multiple EBMs can be seamlessly combined to solve tasks far more complex than those any individual model was trained on [25], [27], [29]. For example, planning in a new environment can be formulated as optimizing over a candidate trajectory that jointly minimizes (i) an energy function capturing whether the trajectory obeys the underlying dynamics and (ii) another energy function verifying goal satisfaction [41]. This modular "plug-and-play" structure allows EBMs to be programmed much like software components. ❷ **Adaptability**. The energy function provides a flexible interface for injecting arbitrary behavioral constraints, task requirements, or structural priors. Thus, EBMs can be repurposed across highly diverse domains without retraining from scratch. For instance, by composing energy terms corresponding to individual logical clauses, EBMs can systematically tackle challenging SAT-style reasoning tasks [27]. Similarly, they can generalize to novel physical-interaction scenarios (e.g., new object manipulations) by encoding previously unseen physical constraints directly into the energy landscape [30]. ❸ **Reasoning Ability**. EBMs employ iterative sampling and energy minimization procedures, known as MCMC (Markov Chain Monte Carlo), that naturally support multi-step refinement, enabling a form of deliberative, System-2-like reasoning. This iterative refinement allows the model to adapt its computation to each new task instance, granting strong performance on tasks requiring multi-hop, zero-shot, or compositional reasoning. For example, [24] frames reasoning itself as repeated energy minimization and demonstrates exceptional results on complex out-of-distribution benchmarks.

However, these benefits come with substantial computational cost. Despite their strong reasoning and generative capabilities, EBMs remain notoriously difficult to deploy efficiently in practical systems, from large-scale cloud services to real-time robotics and edge platforms. As shown in TABLE I, EBMs can exhibit inference latency that is hundreds of times higher than that of mainstream neural models, due to the repeated forward and backward passes required for sampling and optimization. Such system-level inefficiency also makes it impractical to incorporate large-scale ML models, such as Transformers with billions of parameters, into the EBM framework.

Training is even more expensive: the long MCMC chains needed for stable learning lead to high variance, slow convergence, and considerable instability, which significantly hinders widespread adoption. Besides, the model and operation heterogeneity across EBM-based architectures further compounds the challenge. Different energy components often exhibit distinct computational patterns, memory footprints, and dependency structures. When combined into a single compositional model, this diversity results in highly irregular workloads that are poorly matched to existing hardware accelerators, causing severe under-utilization and inefficiency across hardware computing platforms.

Tasks	Reasoning [24]	Compositional Reasoning [37]	Image Generation [36]	Compositional Image Generation [23]	Scene Understanding [28]
EBM accuracy over SOTA	+4.80%	+33.00%	+31.80%	+18.50%	+12.50%
EBM latency over SOTA	20×	380×	250×	330×	420×

TABLE I: Algorithm and system performance comparisons between EBMs and conventional ML models.

Thus, as illustrated in Fig. 1, to systematically understand the system-level challenges of EBM, in this work, we perform an extensive categorization and characterization of EBM workloads, spanning a broad spectrum of representative algorithms currently explored in this area. In addition, we propose cross-layer optimization solutions to improve EBM computing performance, efficiency and scalability, thereby pinpointing the challenges in real-world EBM deployment and outline future directions of EBM research. In conclusion, we make the following contributions:

- We create a comprehensive collection of EBM workloads, systematically summarize their implementations and applications, and propose a unifying taxonomy grounded in their algorithmic structures and system-level implications. (Sec. III & Sec. IV)
- We formulate the system performance of EBMs and conduct extensive profiling and characterization of their algorithmic and system performances across modern computing platforms. (Sec. V)
- We identify and propose potential optimization opprotunities at the algorithm, system and hardware levels to guide future research towards efficient and scalable EBM systems. (Sec. VI)

To the best of our knowledge, this is the ***first work*** *to assess EBM computing from both system and architecture perspectives*, aiming to inspire the design and research of next-generation generative and reasoning systems through synergistic advancements in EBM.

II. EBM Algorithm Structure and Features

EBMs have emerged as a flexible class of ML frameworks capable of capturing arbitrary data distributions and enabling compositional reasoning and generation. In this section, we introduce the core concepts behind EBMs and outline the algorithmic structure of both inference and training, establishing the foundation for later system-level analysis. Key terminologies are highlighted in *italic*.

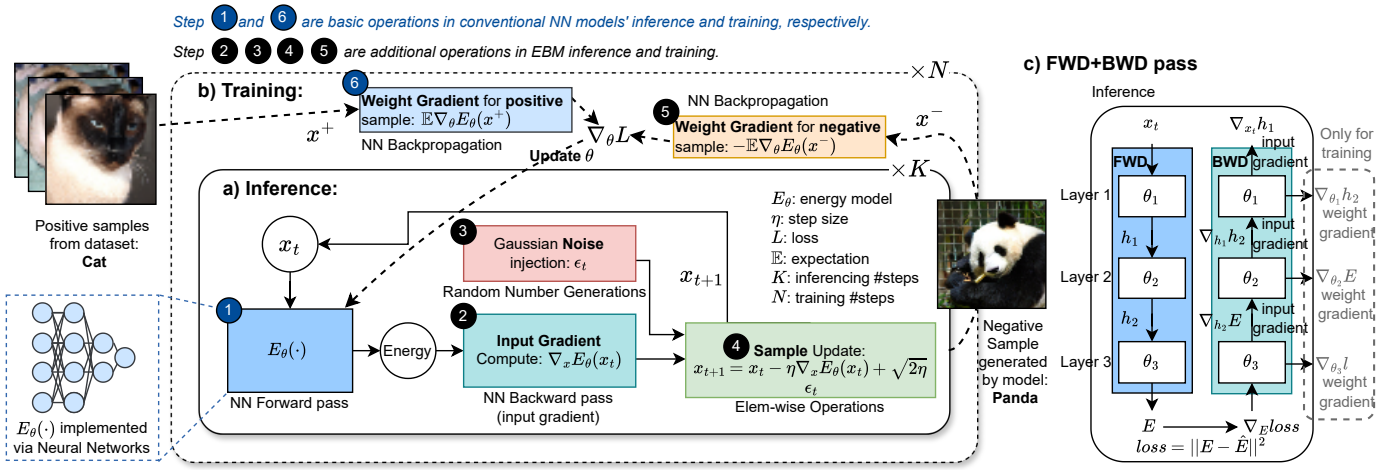


Fig. 2: EBM algorithm architecture and operations. (a) and (b) illustrate generic EBM inference and training algorithm architecture, with an example task to generate a horse image. (c) showcases forward and backward pass operations for EBM inference where backward only computes input gradients, to discriminate with training backpropagation which computes both input and weight gradients.

A. EBM Concepts and Algorithm Architecture

EBMs define an unnormalized probability distribution $p_\theta(x)$ over data using an *energy function* $E_\theta(\cdot)$ implemented by an *energy model*, typically a neural network parameterized by θ that learns the global *energy landscape*. For any input x, the model assigns a scalar energy $E_\theta(x)$, where lower values correspond to higher compatibility or likelihood. The induced distribution is

$$p_\theta(x) \propto e^{-E_\theta(x)} \tag{1}$$

EBM inference seeks an x that minimizes this *energy function*, i.e., region lying at the bottom of the learned *energy landscape*. This is achieved through *Markov Chain Monte Carlo (MCMC)* sampling, which iteratively transforms an initial random sample into one that better aligns with the target distribution. Popular MCMC variants include *Langevin Dynamics* [42], Metropolis-Adjusted Langevin Dynamics (MALA) [43], and Hamiltonian Monte Carlo (HMC) [44]. For illustration, we use *Langevin Dynamics*, a widely adopted method in EBM [40]. Each update step is defined as

$$x_{t+1} = x_t - \eta\,\nabla_x E_\theta(x_t) + \sqrt{2\eta}\,\epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I). \tag{2}$$

where x_t and x_{t+1} denote the current and next samples, η is the step size, $\nabla_x E_\theta(x_t)$ is the gradient of the *energy function* with respect to the input, and ϵ_t is injected Gaussian noise encouraging exploration.

Fig. 2(a) illustrates the four core steps of EBM inference within a single iteration at timestep t: ❶ the current sample x_t is fed through the *energy model* $E_\theta(\cdot)$ to obtain its energy $E_\theta(x_t)$ via a forward pass; ❷ a backward pass computes the *input gradient* $\nabla_x E_\theta(x_t)$; ❸ Gaussian *noise* ϵ_t is generated for stochastic exploration; and ❹ the sample is updated to produce x_{t+1} based on the Langevin update rule. The dimensionality of x_t is task-dependent—for instance, pixel space for image generation, or latent/embedding variables for planning and reasoning. These four steps repeat for K iterations of MCMC sampling, where K is typically chosen empirically for each model and task in existing EBM works.

EBM Training optimizes model parameters θ such that the induced distribution $p_\theta(x)$ matches the true data distribution $p_D(x)$. A widely used approach is contrastive divergence [40], [45], which decreases the energy of real samples while increasing that of model-generated samples:

$$\nabla_\theta L = \mathbb{E}_{x^+\sim p_D}\left[\nabla_\theta E_\theta(x^+)\right] - \mathbb{E}_{x^-\sim p_\theta}\left[\nabla_\theta E_\theta(x^-)\right]. \tag{3}$$

Here, x^+ denotes a positive sample drawn from the data, and x^- is a negative sample obtained through MCMC sampling under the current model. The gradients $\nabla_\theta E_\theta(x)$ drive the maximum-likelihood update of θ.

As visualized in Fig. 2(b), training consists of: (i) executing K MCMC steps for generating each negative sample; (ii) two backpropagations to compute parameter gradients for x^- and x^+ (Steps ❺ and ❻); and (iii) updating the weights of the *energy model* for each of the N training steps. In practice, the K used during training is much smaller than that used at inference time. This is because training only requires negative samples that approximate the model distribution well enough for contrastive divergence, whereas inference typically demands a fully mixed Markov chain for high-quality predictions and generations.

Despite variation in implementations and components across different EBMs, all methods follow this same algorithmic template as discussed in the next section.

B. EBM Algorithm Features

Forward + Backward Pass. Unlike standard NN inference, which runs a single forward pass per input, inference in EBMs repeatedly performs a forward pass of the *energy model* followed by a backward pass that computes only the *input gradient*, no weight gradients are involved, creating

substantial latency. Fig. 2(c) illustrates this with a simple three-layer network: the sample x_t is taken as input for forward pass to produce energy E, then gradients w.r.t. the input are then backpropagated through all layers, where weight-gradient computation (computed only during training) is excluded.

MCMC Sampling. Inference relies on MCMC sampling method to navigate the learned *energy landscape*, primarily governed by the *step size* and the *number of sampling steps*. Smaller steps and longer chains typically improve sample quality but increase runtime linearly. Determining the right balance is nontrivial, yet most existing EBM implementations tune these hyperparameters manually [23], [37], [40], often resulting in suboptimal accuracy-efficiency trade-offs.

Compositionality. A key strength of EBMs is that independently trained *energy functions* can be combined to form richer *energy landscapes* without additional retraining. This enables powerful compositional generation and reasoning capabilities [25]. For example, [37] synthesizes a smiling, wavy-haired male face by merging separate energy models trained on each attribute. However, such compositional designs introduce heterogeneity across architectures and kernels, causing significant inefficiency in real-system deployments.

The following sections are structured as follows: Sec. III proposes our taxonomy categorizing around twenty selected EBM workloads, and showcases EBM algorithm designs with selected models. Sec. IV formulates EBM's runtime performance to help understand profiling results and system bottlenecks. Sec. V profiles and characterizes their system behavior to uncover the cause of EBM inefficiencies, and reports the MCMC sampling impacts on system and algorithmic performances. Sec. VI discusses possible optimizations and future research directions for efficient EBM deployment.

III. EBM Taxonomy and Workloads

EBMs increasingly combine diverse neural network architectures or sometimes symbolic components to define complex energy landscapes to address compositional generation and higher-order reasoning. Consequently, these paradigms can be classified by how their constituent modules are integrated and the resulting dataflow characteristics. In this section, we propose a structural taxonomy for EBM compositionality to characterize model heterogeneity (Sec. III-A), followed by an analysis of data dependencies to elucidate the system-level implications of EBM execution (Sec. III-B). Our proposed taxonomy not only clarifies the architectural attributes of current EBMs but also provides a roadmap for optimizing hardware and software stacks for EBM workloads. We select around twenty representative EBM workloads to form a comprehensive collection to capture the full diversity and broad applicability of EBMs.

A. EBM Compositionality Taxonomy

We primarily categorize EBMs into three distinct classes based on their architectural compositionality in TABLE II(a).

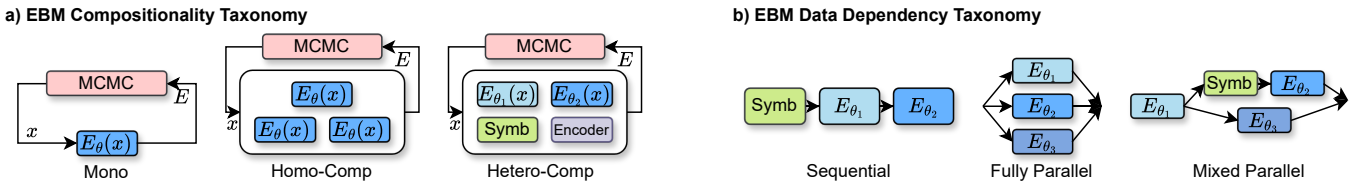
*1) **Monolithic (Mono)**:* This category represents the fundamental EBM implementation, utilizing a single Neural Network to approximate the energy function without external algorithmic integration. During inference, these models invoke a solitary FWD+BWD pass per step, resulting in a linear computation pattern. For instance, EBT [26] constructs an energy function using a transformer architecture, enabling "System 2" reasoning capabilities that demonstrate superior out-of-distribution performance compared to standard diffusion language models.

*2) **Homogeneous Compositional (Homo-Comp)**:* EBMs in this category construct the energy landscape with multiple instances of an identical neural architecture. During sampling it applies the same energy model to various inputs or features. Thus despite the compostionality multiple NN instances bring, the operation kernels remain uniform. We further differentiate this category based on their weight-sharing characteristics:

Homo-Comp-DS (Duplicated Structure). In this subtype, NN instances are trained independently to capture different features, resulting in different weights and parameters for each instance which precludes weight reuse during MCMC iteration. For example, EBM-COMP [37] trains separate ResNet-like networks for specific attributes (e.g., gender, expression, hair color) and subsequently merges their energy landscapes via logical conjunction, disjunction, and negation to compose facial imagery.

Homo-Comp-DW (Duplicated Weights). Conversely, this subtype runs exactly the same NNs with same weights for all inputs and features, creating significant opportunities for data reuse and cache locality across consecutive forward and backward passes. To discriminate from *Mono* models, these systems execute multiple instances of the same model—either sequentially or in parallel—to extract distinct problem features, thereby demanding different runtime parallelism strategies for efficiency. EBM-GR [27] utilizes this approach for reasoning, i.e. in the N-Queens problem, a single MLP-based diffusion model is trained to validate a row but is reused simultaneously across all rows, columns, and diagonals during inference. Similarly, IRED [24] employs an MLP to learn annealed energy landscapes within a generalized framework for reasoning and planning tasks.

*3) **Heterogeneous Compositional (Hetero-Comp)**:* The most complex class of workloads, Hetero-Comp, integrates disparate neural network architectures to form the energy model, sometimes coordinated with symbolic components (e.g., logical encoders, replay buffers, constraint solvers). This heterogeneity supports cohesive problem-solving across generative and cognitive domains. For instance, EBM-CVR [36] develops relational energy functions by leveraging CLIP [49] for symbolic scene encoding, while utilizing distinct architectures to compute energy for object features versus spatial relations. In the reinforcement learning domain, EBM-MBP [34] combines an online learning algorithm with a replay buffer, allowing the model to store trajectories and plan via continuous interaction with the environment. Furthermore, CCSP [30] addresses continuous constraint satisfaction prob-



Compositionality		Data Dependency	Work	Year	Models	MCMC	Tasks
Mono		Sequential	EBM-IMP [40]	2019	CNN	Langevin Dynamics	Image generation
			EBM-EqM [46]	2025	Transformer-based NN	Langevin Dynamics	Image generation, OOD detection
			EBT [26]	2025	Transformer	Langevin Dynamics	System 2 thinking, image denoising
Homo-Comp	DW	Mixed Parallel	IREM [31]	2022	GNN, MLP	Langevin Dynamics	Reasoning, planning
			IRED [24]	2024	GNN, MLP	Langevin Dynamics	Reasoning, planning
			EBM-GR [27]	2025	MLP based Diffusion Model	Langevin Dynamics	Reasoning, Continual Learning
		Fully Parallel	RRR [23]	2024	U-Net based Diffusion Model	Langevin Dynamics, MALA, HMC	Compositional image generation
	DS		EBM-CMP [37]	2020	CNN	Langevin Dynamics	Image generation
			EBM-IGM [28]	2025	Diffusion model	Langevin Dynamics	Scene understanding
Hetero-Comp		Sequential	ADE [39]	2019	CNN, Deep LVM	HMC	Maximum Likelihood Estimation (MLE)
			EBM-EBP [47]	2021	MLP, Replay Buffer	Langevin Dynamics, HMC	Planning, Reinforcement Learning
		Mixed Parallel	EBM-CDT [38]	2021	Multi-scale CNNs	Langevin Dynamics	Compositional image generation, OOD detection
			EBM-CVR [36]	2021	Multi-scale CNNs, CLIP	Langevin Dynamics	Image generation/editing/classification, scene understanding
			EBM-CL [32]	2022	MLP, CNN	Langevin Dynamics	Classification w/ continual learning
			COMET [35]	2021	CNN, Convolutional Encoder	Langevin Dynamics	Unsupervised scene understanding
		Fully Parallel	EBM-CVG [33]	2023	U-Net based Diffusion Models	Langevin Dynamics	Image generation
			CCSP [30]	2023	MLP-based Diffusion Model, Transformer Diffusion Model	Langevin Dynamics	Constraint Solving
			EBM-CGID [48]	2024	Diffusion model (U-Net)	Langevin Dynamics	Inverse Design

TABLE II: EBM Taxonomy. Typical EBM workloads categorized by i) Compositionality, ii) Data Dependency (Sec. III). Each category highlights representative algorithms with their models employed, MCMC sampling methods, and targeting tasks.

lems in robotics [50]–[54] by aggregating energies from diffusion models trained on individual constraints, demonstrating robust generalization to novel constraint combinations.

B. EBM Data Dependency Categorization

After defining EBM's structural components, we categorize their data dependencies to identify parallelism potential and provide efficient scheduling implications. As illustrated in Table II(b), we identify three dependency patterns:

Sequential. This type enforces a sequential execution order within each sampling step due to strict data dependencies between models. While Mono models are certainly sequential, some Hetero-Comp models also fall into this type. For example, ADE [34] uses a deep Gaussian latent variable model (Deep LVM) to initialize distributions and produce outcomes to execute the CNN-based discriminator's HMC steps.

Fully Parallel. In this paradigm, energy models and components exhibit zero inter-dependency, enabling maximal and flexible runtime parallelism. A Hetero-Comp example is COMET [35], which utilizes separate ResNet-based energy models for concepts and a convolutional encoder for latent factor inference; all components consume the same image input and can execute simultaneously. The majority of Homo-Comp workloads also align with this high-throughput category.

Mixed Parallel. These workloads present hybrid dependencies consisting of both parallel branches and sequential executions. EBM-CDT [38] for example, uses a multi-scale architecture where the input image is down-sampled to various levels. While different levels can be processed in parallel, the models and operations within each level are enforced to run in sequential.

Each category of EBM systems exhibits distinct kernel operators, data dependencies, and compositional structures. *This work presents the first system-level effort to characterize EBM's computational behavior, laying the foundation for the design and deployment of both current and future EBM systems.*

IV. EBM Performance Formulation

In this section we propose our formulation for EBM's runtime to identify system bottleneck implications and help with understanding profiling results presented in next section.

In light of the EBM algorithm presented in Fig. 2(a), we first formulate the runtime of EBM's one *sampling step*:

$$T_{samp} = T_{fwd} + T_{bwd,in} + T_{rng} + T_{update} \quad (4)$$

T_{fwd} and $T_{bwd,in}$ denotes the latency of one forward pass in step ❶ and one input backward pass in step ❷, respectively. T_{rng} represents the runtime of random noise generation for

step ❸, and T_{update} is the latency of updating the sample in step ❹. Due to the small runtime percentages these two steps exhibit observed in Sec. V, we further omit the latency of both noise generation and sample update for mathematical simplicity:

$$T_{samp} \approx T_{fwd} + T_{bwd,in} \tag{5}$$

This is because typically the generation of noise can be parallelized with backward pass (step ❷) using a group of random number generators (RNG) with much less latency, while sample update is just an element-wise operation at sample dimension that can be easily computed in SIMD units with ignorable latency.

Then we derive EBM inference runtime T_{inf} as:

$$T_{inf} \approx T_{samp} \times C \times K \tag{6}$$

where C is the number of MCMC chains, normally equal to batch size; K represents number of MCMC sampling steps in inference. Note that this formulation aims to capture standard EBM runtime in a general sense, thus additional non-energy-model components or trivial operators in specific workloads (e.g. symbolic programs, encoders) are excluded from this analysis. Also, here we assume sequential MCMC sampling executions.

Based on the training steps presented in Fig. 2(b), we define training runtime T_{train} as:

$$T_{train} \approx N \times (T_{bwd,w}(x^+) + T_{bwd,w}(x^-) + T_{samp} \times C \times K_t) \tag{7}$$

where N is the number of training iterations; K_t is the number of MCMC steps used for generating negative samples, typically fewer than the inference sampling steps K, as discussed in Sec. II-A; $T_{bwd,w}(x^+)$ and $T_{bwd,w}(x^-)$ denote the backpropagation times for computing weight gradients using positive samples x^+ from data and negative samples x^- generated through K_t-step inference, respectively. Thus, each training step includes two weight gradient backpropagations (steps ❺ and ❻) and one inference procedure to generate (steps ❶ and ❷ repeated K times). As shown in Sec. V, the inference portion $T_{samp} \times C \times K_t$ still dominates the overall training runtime in practice.

V. Workload Characterization Results

In this section we present the profiling results and analyzes the performance characteristics of representative EBM workloads covering our entire taxonomy spectrum.

A. Profiling Methodology

System Setup. For *system-level* characterization, we first conduct function-level and kernel-level profiling to capture statistics such as runtime, memory, tensor sizes of each model by leveraging the built-in PyTorch Profiler [55]. For those workloads written in other ML frameworks (i.e. Tensorflow) we re-implement them in PyTorch to ensure system consistency and fair comparison. We also perform post-processing to partition the characterization results into various operation categories. The experiments are conducted on Ryzen 9900X CPU, Nvidia RTX 5090 GPU, and Google TPU v6e.

Workload	Compositionality	Data Dependency	Models	Size	Intensity	Steps
EBM-IMP [40]	Mono	Sequential	CNN	30 MB	55 FLOPs/Byte	100
EBM-EqM [46]	Mono	Sequential	Transformer	2.7 GB	175 FLOPs/Byte	250
IRED [24]	Homo-Comp-DW	Fully Parallel	MLP	31 MB	100 FLOPs/Byte	50
EBM-CMP [37]	Homo-Comp-DS	Fully Parallel	Duplicated CNNs	56 MB	119 FLOPs/Byte	400
EBM-CVR [36]	Hetero-Comp	Mixed Parallel	CNN 128x128	15 MB	150 FLOPs/Byte	300
			CNN 64x64	3.8 MB	125 FLOPs/Byte	
			CNN 32x32	1 MB	94 FLOPs/Byte	
			CLIP	252 MB	23 FLOPs/Byte	
CCSP [30]	Hetero-Comp	Fully Parallel	MLP	12 MB	42 FLOPS/Byte	1000
			CNN Encoder	23 MB	20 FLOPS/Byte	

TABLE III: Profiled Workloads. Six profiled workloads with their model compositionality and data dependency categories, model types and sizes, arithmetic intensities, and MCMC sampling steps.

Algorithm Setup. For a complete evaluation across our entire compositionality and data dependency taxonomy spectrum defined in Sec. III, we pick six workloads listed in TABLE III. We select EBM-IMP and EBM-EqM for *Mono* models. Among *Homo-Comp* models we pick IRED (*DW & Mixed Parallel*) and EBM-CMP (*DS & Fully Parallel*). In *Hetero-Comp* models we use EBM-CVR (*Mixed Parallel*) and CCSP (*Fully Parallel*). For algorithm setup and evaluation we utilize the workloads' original dataset including MNIST [15], CIFAR [56], CLEVR [57], SAT-Net [58], CelebA [59]. to study the impact of sampling on their accuracy.

B. Compute Latency Analysis

End-to-end runtime. We first characterize the end-to-end runtime of representative EBM workloads on modern CPU (Ryzen 9900X), GPU (RTX 5090) and TPU (v6e). Fig. 3(a) reports the latency of a *single* MCMC sampling step, corresponding to inference Steps ❶–❹. We can observe that the average runtime of a sampling step is at millisecond level on GPU and CPU, comparable to conventional NNs due to the neural operation in FWD+BWD pass (Step ❶ and ❷) being the core of the execution as specified in Equation 5. However, during inference, this sampling step is repeated extensively throughout the MCMC process. As a result, the total inference latency can scale hundreds or even thousand of times as shown in Fig. 3(b), reaching tens of seconds even on high-end GPUs and TPUs. Such latency is clearly incompatible with real-time or interactive deployment scenarios. In addition, we can see that for both single-step sampling and full inference, GPUs and TPUs achieve up to two-orders-of-magnitude speedups over CPUs. This behavior stems from the fact that most EBM kernels remain compute-intensive and highly parallelizable, closely resembling conventional neural network workloads. In contrast, smaller models with lower arithmetic intensity, such as EBM-IMP and CCSP, exhibit less performance gain on GPUs and TPUs over CPUs, further confirming this point.

Takeaway 1: *EBMs typically exhibit similar system performance and characteristics as conventional neural models in one sampling step, but exhibit much higher latency during inference due to heavy sampling repetition, prohibiting them from real-time applications.*

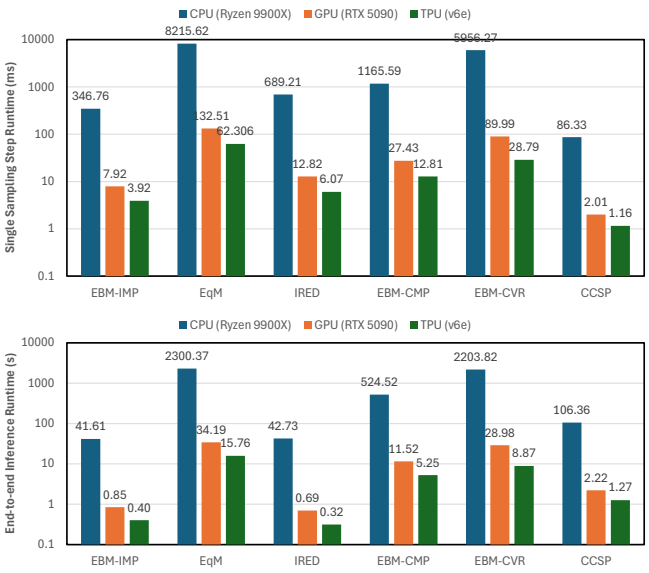


Fig. 3: EBM End-to-End Runtime. (a) Runtime of one sampling step in **millisecond**. (b) Runtime of inference procedure in **second**.

End-to-end runtime breakdown. Fig. 4 presents the end-to-end latency breakdowns of representative EBM workloads during inference and training. Inference is evaluated on CPU, TPU and GPU, while training results are reported on GPU only. In Fig. 4(a) we can conclude that (1) As the main bottleneck, FWD+BWD passes (Step ❶ and ❷) dominate the overall runtime, accounting for up to 99% of inference time across all workloads and platforms, while the runtime of random noise generation in Step ❸ and sample update in Step ❹ are nearly ignorable, validating the approximation adopted in our performance formulation in Equation 5. (2) The latencies of the FWD+BWD passes are highly comparable, because the input gradient computation in backward pass is nothing but reversed matrix multiplications of fwd pass with transposed weights, incurring similar operation kernels and scheduling patterns. (3) FWD+BWD passes cannot be parallelized due to strict data dependencies between these two steps, doubling the runtime of neural network execution.

Fig. 4(b) presents the runtime breakdown during training, showing that (1) Up to 90% of training time is still dominated by MCMC sampling process where each stage exhibits same percentage as in inference. (2) Depending on the number of sampling steps during training, the weight gradient descent and parameter update in Step ❺ and ❻ can take up to 30% of the total training time, creating additional optimization opportunities. (3) Compared to training stages (Steps ❺ and ❻), the inference latency of EBMs scales more rapidly with model compositionality and size, as shown in Fig. 4(b), indicating that accelerating MCMC sampling chain with heavy NN FWD+BWD pass iteration is the key to improving performance and scalability of EBM systems for both training and inference.

Takeaway 2: *The iterative forward and backward pass (Step ❶ and ❷) dominate runtime across all EBM algorithms in both inference and training, exhibiting comparable latency and strong intra/inter-iteration data dependencies.*

C. Model and Operator Heterogeneity

System Roofline Analysis. Fig. 5 employs the roofline model to quantify the compute and memory boundedness of the selected EBM workloads on Nvidia RTX 5090 GPU. For workloads consisting of various energy models implemented via different networks, we plot the components separately in the roofline figure. We can observe that while different EBM workloads often exhibit diverse memory and compute characteristic, the energy models implemented by a single EBM - especially *Hetero-Comp* models - can also cover a wide range of arithmetic intensity, inducing substantial operational heterogeneity that causes significant system inefficiency and performance degradation. For example, EBM-CVR (purple) employs compute bounded ResNets at different image down-sampling level with operational intensity ranging across 60 FLOPs/Byte, while having a memory bounded transformer-based model CLIP for text understanding. Meanwhile, all of CCSP's energy model are heavily memory bounded.

Operator Analysis. Fig. 6 breaks down the runtime of EBM workloads into five operator categories, revealing a severe degree of kernel heterogeneity that distinguishes EBMs from conventional ML workloads. While standard models are often uniformly dominated by matrix multiplications (MatMul) or convolutions, EBMs exhibit a distinct bifurcation: EqM and IRED are heavily MatMul-oriented (72.9% and 33.7% respectively), whereas EBM-CMP, EBM-IMP, and EBM-CVR remain Convolution-dominant (50%-69%). Crucially, EBMs suffer from significant non-compute bottlenecks absent in standard inference: IRED spends 32.8% of its runtime on Data Transformation, driven by the need to concatenate features during iterative reasoning steps. Similarly, EBM-CVR spends 45.3% of time on Vector/Element-wise operations, while CCSP and EBM-IMP devote around 30% to Data Movement, highlighting how the iterative MCMC sampling process exposes severe overheads in memory-bound operations that accelerator tensor cores cannot optimize.

Takeaway 3: *EBMs exhibit significant diversity across all workloads, with high model and operator heterogeneity in each workload (especially Hetero-Comp models), spanning compute-bound and memory-bound regimes, causing significant accelerator under-utilization and system inefficiency.*

D. MCMC Sampling Study

Here we study the impact of MCMC sampling procedure - the skeleton of EBM's inference and training - on both system performance scaling and algorithmic accuracy. To help further understand EBM efficiency challenges and identify system and algorithm optimization opportunities and directions.

Performance Scaling Fig. 7 illustrates how runtime scales with the number of sampling steps for selected workloads. While some workloads incur higher latency in the first sampling step due to substantial model weight loading or data preprocessing, all EBMs exhibit runtime that grows linearly

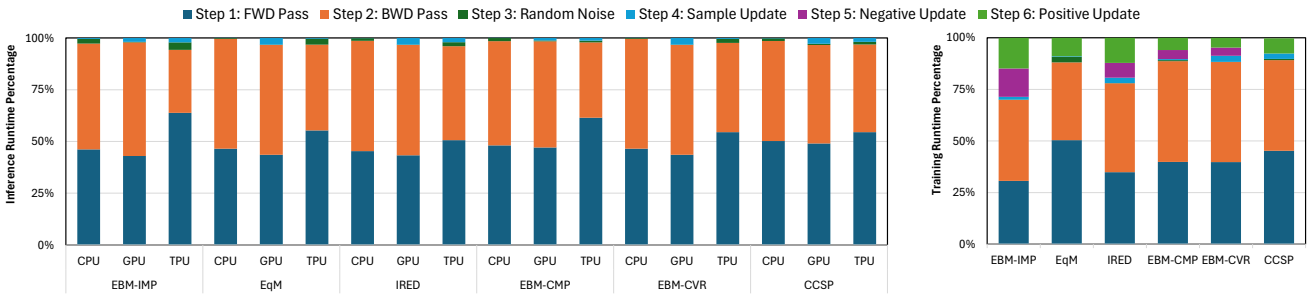


Fig. 4: EBM Runtime Breakdown. (a) Inference latency breakdown on CPU, GPU and TPU, showing FWD and BWD passes being performance bottlenecks, each taking almost half of the runtime across all devices. (b) Training runtime breakdown on GPU, showing Step 1, 2, 5, and 6 as the main runtime cost, where FWD and BWD passes still dominate.

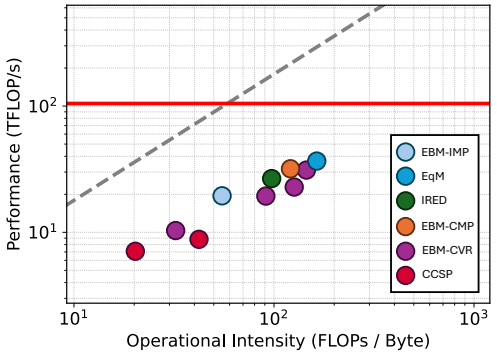


Fig. 5: Roofline Model of the selected workloads on Nvidia RTX 5090. For *Hetero-Comp* models each of the energy models in their implementation is plotted separately.

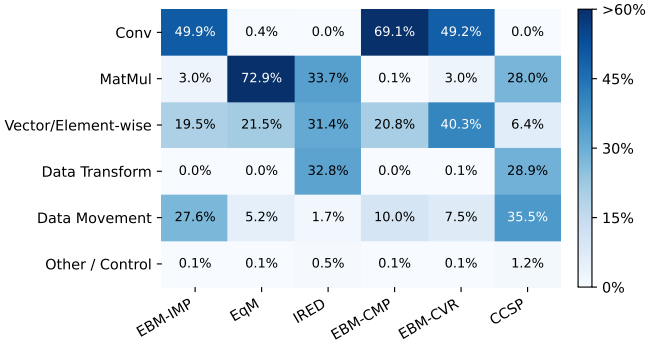


Fig. 6: Operator-level runtime percentage of selected workloads.

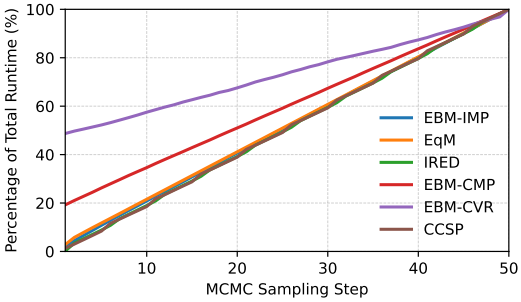


Fig. 7: System performance scaling over number of MCMC sampling steps. Runtime percentage at each step over the total latency of 50 sampling steps during inference of selected workloads, starting from the first step.

with the number of sampling steps, reflecting the inherently iterative nature of both inference and training.

Takeaway 4: *EBM runtime scales almost linearly with number of MCMC sampling steps, causing dominant bottleneck in EBM's performance and scalability.*

Algorithmic Accuracy. Fig. 8 plots the algorithmic performance (accuracy or MSE error) vs number of MCMC sampling steps using three workloads - EqM, IRED (two tasks), and CCSP. In our experiments we set the total number of MCMC sampling steps same in the original algorithm setup, which are selected empirically by the model developers. We have the following observations: (1) The optimal number of MCMC steps that maximizes performance and accuracy differs among EBM workloads. (2) For the same EBM, the optimal sampling steps vary between different tasks. In IRED for example, the continuous algorithmic reasoning task takes around 45 steps to achieve the lowest MSE error (Fig. 8(b)), while 11 steps of sampling already yields the best accuracy for discrete space reasoning task (Fig. 8(c)). Moreover, the substantial accuracy fluctuations observed in CCSP during extended 1000-step sampling further support this point(Fig. 8(d)). Thus, smartly setting the sampling steps for different models and tasks is the key to system and algorithmic efficiency. (3) Each of the sampling step may produce drastically different impact on performance due to the unpredictability of complex energy landscapes that the energy model depicts. For example in Fig. 8(c), step 6 boosts the accuracy to 90% for almost 30%, while the accuracy slightly drops after step 13. Therefore, how to dynamically adjust sampling step sizes to faster search for lowest energy is also a challenge.

Takeaway 5: *Optimal MCMC sampling steps and step sizes vary significantly across EBM workloads, tasks, and even individual iterations due to different model complexity and energy landscapes, highlighting the need for adaptive, model-aware and task-aware sampling schemes to balance accuracy and efficiency.*

VI. Optimization and Research Directions

Based on the above analysis and characterization, in this section, we suggest possible optimization solutions to tackle the challenges summarized in the key takeaways in the above section, guiding future research directions towards efficient EBM algorithms and systems.

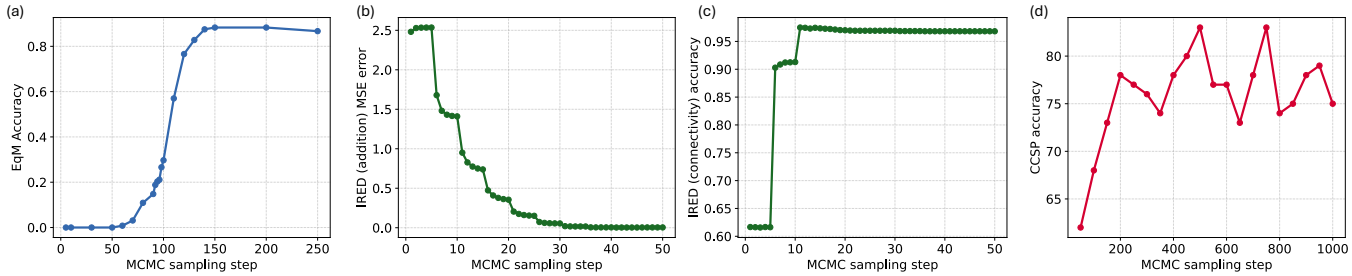


Fig. 8: Algorithmic performance vs number of MCMC sampling steps. (a) EqM accuracy vs sampling steps on ImageNet 256x256 task. (b) IRED MSE error vs MCMC sampling steps on continuous algorithmic reasoning task (matrix addition). (c) IRED accuracy vs MCMC sampling steps on discrete space reasoning task (graph connectivity). (d) CCSP accuracy vs MCMC sampling steps on 2D Shape Arrangement task with Qualitative Constraints.

For Takeaway 1 (NN based iteration):

At algorithm level, typical neural model optimization techniques-such as compression, quantization, and pruning [60]–[64] for neural networks, and KV cache optimization, operator fusion, MoE for transformer models [65]–[69] can still be applied to reduce the computational and memory footprint of energy models. However, it is necessary to explicitly study their unique impact on EBM-specific algorithm behavior, especially regarding algorithmic accuracy in inference, training efficiency, MCMC sampling convergence and energy landscape traversal. This is because (1) While the error brought by such algorithmic optimizations is typically negligible in one execution, it might propagate and cumulate during the long sampling process and cause severe accuracy drop after hundreds of iterations. (2) Compared to explicit neural models that generate output directly from input, EBMs–as implicit models [40]–instead generates scalar energy values from samples, so the same optimizations may have different impact on model behavior and performance. Thus, comprehensive investigation and experiment specifically designed for EBM is needed to ensure that such optimizations do not compromise accuracy and sampling robustness.

To further prove and study this, we apply weight-only post-training quantization (PTQ) schemes to EqM. Since EBMs operate through multi-step sampling, we use a normalized L2-shift metric, rather than accuracy alone, to quantify quantization error at each step:

$$\frac{||O_{quant} - O_{full}||_2}{||O_{full}||_2}$$

where O_{quant} is the quantized model output, and O_{full} is full-precision model output. This metric captures the quantization-induced shift in the energy landscape during inference. We make three key observations in Fig. 9: 1) the shift accumulates over sampling steps, causing increasing divergence from the full-precision model, and 2) For EqM, FP8 is the most robust format, while INT8 and FP4 are much less stable. In particular, INT8 shows even larger and more erratic shifts than NF4 across consecutive steps. 3) At 4-bit precision, the optimal quantization scheme varies with the number of steps. This demonstrates the need for customized and comprehensive quantization strategies tailored to specific EBM algorithms and deployment settings.

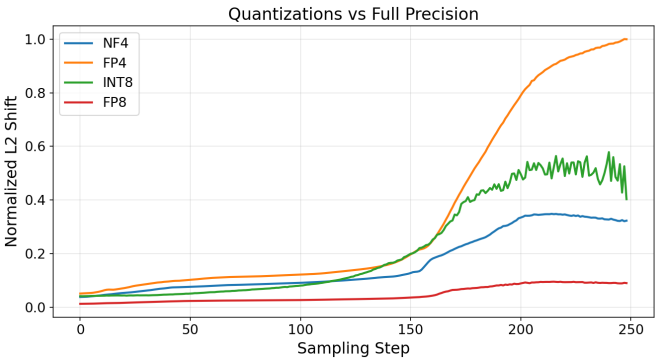


Fig. 9: Quantization impact on EBM inference. Using EqM as a case study, the results show that errors introduced by different PTQ schemes accumulate differently across EBM sampling steps.

At system level, existing AI accelerator designs [70]–[74] may be leveraged due to the strong alignment between EBM kernels and typical neural model operators. Nonetheless, EBM-specific features such as repeated back-to-back FWD+BWD passes, iterative MCMC sampling loops, model compositionality, and operator heterogeneity introduce unique system requirements, necessitating customized hardware designs and unique scheduling strategies to facilitate the deployment and development of EBMs.

For Takeaway 2 (repetitive FWD+BWD pass):

At algorithm level, the challenge and opportunity is to reduce the cost of each sampling step by either relaxing the strict data dependencies between forward and backward passes, or eliminating unnecessary computations. This may be achieved via novel approximation techniques to exploit parallelism between FWD and BWD passes, sampling step skipping or selectively reducing gradient computation when energy descent stabilizes to reduce per-step runtime while maintaining convergence quality. For example, prior work has proposed methods to parallelize stochastic gradient descent (SGD) in neural network training [75], [76], which may inspire the development of analogous—but distinct—approaches to break dependencies and enable parallelism in EBM sampling.

At system level, the iterative nature of EBM inference results in frequent reuse of the same weights, as both FWD and BWD passes operate on identical parameters (with BWD using their transposed form). Efficient memory system design

enabling reuse of weights across back-to-back passes opens up opportunities for unified compute pipelines, weight sharing mechanisms, and low-overhead transposition support. Moreover, when iterative FWD+BWD execution is combined with compositional models, the resulting irregular compute patterns and inter-model dependencies pose additional challenges for architecture and scheduling design.

For Takeaway 3 (compositionality and heterogeneity): *At algorithm level*, optimizing the model size, complexity, and heterogeneity and implementing more hardware-friendly models can directly improve system performance and reduce acceleration pressure. In addition, exposing model-level parallelism within compositional algorithm architectures can enable more efficient scheduling and execution, particularly for workloads with independent or loosely coupled components.

At system level, efficient support for EBMs requires hardware system that can accommodate a mix of compute-bound and memory-bound kernels, especially for compositional models. Architectures must be flexible enough to adapt to shifting bottlenecks across heterogeneous components, while scheduling policies should dynamically balance different operators to maximize resource utilization and maintain system throughput. We demonstrate this on compositional workloads (IRED, EBM-CMP, EBM-CVR, and CCSP) by launching each energy model in a separate CUDA stream to enable model-level parallelism. Even on a single GPU, this exposes kernel overlap and achieves speedups of 1.28×, 1.47×, 1.21×, and 1.32×, respectively. Greater performance gains are possible with EBM-aware hardware and scheduling.

For Takeaway 4 & 5 (MCMC sampling): Enhancing the efficiency of MCMC sampling, through shorter sampling lengths, faster steps, and quicker convergence, without compromising algorithmic accuracy is the key to improving EBM runtime. While prior work has explored general MCMC optimization and acceleration [44], [77]–[79], these efforts have not addressed the specific context of EBMs, where sampling is realized through FWD+BWD passes using NN kernels as the major operations. Thus the same algorithmic and architectural optimization can barely benefit EBM inference and training. Moreover, as demonstrated in Fig. 8, different EBM models and tasks necessitates varying sampling lengths as well as convergence criteria.

To address this, one promising direction is to develop algorithms that smartly determine the sampling length either statically at compile time or dynamically at runtime, adapting to the task and model behavior. Additionally, adjusting the sampling step size in response to the local energy landscape can accelerate convergence by avoiding flat or oscillatory regions. Finally, runtime convergence detection mechanisms can be employed to terminate sampling early, thus avoiding unnecessary sampling steps as observed in Fig. 8, and thus reducing overall latency. Compared to the sampling schemes implemented in most of the existing EBMs where MCMC sampling length is determined empirically, a combination of above methodologies can bring substantial runtime reduction given that latency scales linearly with MCMC sampling steps, achieving efficient real-time EBM deployment and training.

Step size	0.25×	0.5×	1×	2×	4×	8×	Adaptive sampling
Number of sampling steps	1038	722	494	**351**	610	911	**170**
Optimal accuracy	79%	**84%**	82%	79%	58%	7%	**86%**

TABLE IV: Adaptive sampling performance showcase on CCSP.

We show the feasibility and necessity of adaptive sampling by evaluating step-size scaling and convergence criteria on CCSP and EqM (TABLE IV). We measure both total sampling steps and best accuracy under step sizes ranging from 0.25× to 8× the default. For CCSP, we use a simple adaptive schedule: 4× until 50% accuracy, 2× until 70%, 1× until 80%, and 0.5× until stopping. Stopping is determined when:

$$\frac{\|n_t\|}{\|\operatorname{avg}(g_{t-10:t})\|} > \alpha$$

where n_t is the noise at step t, g_t is the step-t gradient, and $\operatorname{avg}(g_{t-10:t})$ is the average gradient over the last 10 steps; $\alpha = 0.1$ for CCSP. TABLE IV shows that this simple adaptive method significantly reduces sampling steps while also improving accuracy, highlighting the need for EBM algorithm-specific sampling optimizations.

VII. Conclusion

EBM is an emerging paradigm for next-generation robust, explainable, and compositional cognitive AI Systems. This paper comprehensively collects and categorizes EBM workloads, systematically formulates and characterizes EBM's system performance, analyzes their workload operators and system bottlenecks, and proposes optimization techniques for both system-level and algorithm-level performance and efficiency, identifying the challenges and opportunities towards fulfilling future System-2 AI systems.

Acknowledgments

We thank Ananda Samajdar for technical discussions, Akshat Ramachandran for helping with data formatting, and the anonymous ISPASS reviewers for their thorough comments and feedback. This work was supported in part by CoCoSys, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] Y. Yan, Y. Lu, R. Xu, and Z. Lan, "Do phd-level llms truly grasp elementary addition? probing rule learning vs. memorization in large language models," *arXiv preprint arXiv:2504.05262*, 2025.

[2] C. Qian, P. Han, Q. Luo, B. He, X. Chen, Y. Zhang, H. Du, J. Yao, X. Yang, D. Zhang, *et al.*, "Escapebench: Pushing language models to think outside the box," *arXiv e-prints*, pp. arXiv–2412, 2024.

[3] D. Kahneman, *Thinking, fast and slow*. macmillan, 2011.

[4] Z. Wan, H. Yang, R. Raj, C.-K. Liu, A. Samajdar, A. Raychowdhury, and T. Krishna, "Cogsys: Efficient and scalable neurosymbolic cognition system via algorithm-hardware co-design," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 775–789, IEEE, 2025.

[5] Z. Wan, C.-K. Liu, H. Yang, R. Raj, C. Li, H. You, Y. Fu, C. Wan, A. Samajdar, Y. C. Lin, *et al.*, "Towards cognitive ai systems: Workload and characterization of neuro-symbolic ai," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 268–279, IEEE, 2024.

[6] Z.-Z. Li, D. Zhang, M.-L. Zhang, J. Zhang, Z. Liu, Y. Yao, H. Xu, J. Zheng, P.-J. Wang, X. Chen, *et al.*, "From system 1 to system 2: A survey of reasoning large language models," *arXiv preprint arXiv:2502.17419*, 2025.

[7] Z. Wan, C.-K. Liu, H. Yang, R. Raj, C. Li, H. You, Y. Fu, C. Wan, S. Li, Y. Kim, *et al.*, "Towards efficient neuro-symbolic ai: From workload characterization to hardware architecture," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, vol. 1, no. 1, pp. 53–68, 2024.

[8] Z. Wan, C.-K. Liu, J. Qian, H. Yang, A. Raychowdhury, and T. Krishna, "Reason: Accelerating probabilistic logical reasoning for scalable neuro-symbolic intelligence," *arXiv preprint arXiv:2601.20784*, 2026.

[9] A. d. Garcez and L. C. Lamb, "Neurosymbolic ai: The 3 rd wave," *Artificial Intelligence Review*, pp. 1–20, 2023.

[10] G. Booch, F. Fabiano, L. Horesh, K. Kate, J. Lenchner, N. Linck, A. Loreggia, K. Murgesan, N. Mattei, F. Rossi, *et al.*, "Thinking fast and slow in ai," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 15042–15046, 2021.

[11] P. Hitzler, A. Eberhart, M. Ebrahimi, M. K. Sarker, and L. Zhou, "Neuro-symbolic approaches in artificial intelligence," *National Science Review*, vol. 9, no. 6, p. nwac035, 2022.

[12] C.-K. Liu, Z. Wan, Y.-S. Noh, M. Ibrahim, S. D. Spetalnick, T. Krishna, W.-S. Khwa, A. S. Lele, Y.-D. Chih, M.-F. Chang, *et al.*, "A 40-nm programmable heterogeneous soc with 5.625/0.85 mb rram/sram for accelerating neuro-symbolic ai models," *IEEE Journal of Solid-State Circuits*, 2026.

[13] K. Hamilton, A. Nayak, B. Božić, and L. Longo, "Is neuro-symbolic ai meeting its promises in natural language processing? a structured review," *Semantic Web*, vol. 15, no. 4, pp. 1265–1306, 2024.

[14] Z. Wan, H. Yang, J. Qian, R. Raj, J. Park, C. Wang, A. Raychowdhury, and T. Krishna, "Compositional ai beyond llms: System implications of neuro-symbolic-probabilistic architectures," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '26, p. 67–84, 2025.

[15] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 2007.

[16] Z. Wan, C.-K. Liu, H. Yang, R. Raj, A. Raychowdhury, and T. Krishna, "Efficient processing of neuro-symbolic ai: A tutorial and cross-layer co-design case study," *Proceedings of the International Conference on Neuro-symbolic Systems*, 2025.

[17] H. Yang, Z. Wan, R. Raj, J. Park, Z. Li, A. Samajdar, A. Raychowdhury, and T. Krishna, "Nsflow: An end-to-end fpga framework with scalable dataflow architecture for neuro-symbolic ai," *arXiv preprint arXiv:2504.19323*, 2025.

[18] M. Ibrahim, Z. Wan, H. Li, P. Panda, T. Krishna, P. Kanerva, Y. Chen, and A. Raychowdhury, "Special session: Neuro-symbolic architecture meets large language models: A memory-centric perspective," in *2024 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pp. 11–20, IEEE, 2024.

[19] S. Du, M. Ibrahim, Z. Wan, L. Zheng, B. Zhao, Z. Fan, C.-K. Liu, T. Krishna, A. Raychowdhury, and H. Li, "Cross-layer design of vector-symbolic computing: Bridging cognition and brain-inspired hardware acceleration," *arXiv preprint arXiv:2508.14245*, 2025.

[20] Y. Yue, Z. Chen, R. Lu, A. Zhao, Z. Wang, S. Song, and G. Huang, "Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?," *arXiv preprint arXiv:2504.13837*, 2025.

[21] A. Karan and Y. Du, "Reasoning with sampling: Your base model is smarter than you think," *arXiv preprint arXiv:2510.14901*, 2025.

[22] Y. Song and D. P. Kingma, "How to train your energy-based models," *arXiv preprint arXiv:2101.03288*, 2021.

[23] Y. Du, C. Durkan, R. Strudel, J. B. Tenenbaum, S. Dieleman, R. Fergus, J. Sohl-Dickstein, A. Doucet, and W. S. Grathwohl, "Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc," in *International conference on machine learning*, pp. 8489–8510, PMLR, 2023.

[24] Y. Du, J. Mao, and J. B. Tenenbaum, "Learning iterative reasoning through energy diffusion," *arXiv preprint arXiv:2406.11179*, 2024.

[25] Y. Du, *Learning Generalizable Systems by Learning Composable Energy Landscapes*. PhD thesis, Massachusetts Institute of Technology, 2025.

[26] A. Gladstone, G. Nanduru, M. M. Islam, P. Han, H. Ha, A. Chadha, Y. Du, H. Ji, J. Li, and T. Iqbal, "Energy-based transformers are scalable learners and thinkers," *arXiv preprint arXiv:2507.02092*, 2025.

[27] A. Oarga and Y. Du, "Generalizable reasoning through compositional energy minimization," *arXiv preprint arXiv:2510.20607*, 2025.

[28] Y. Wang, J. Dauwels, and Y. Du, "Compositional scene understanding through inverse generative modeling," *arXiv preprint arXiv:2505.21780*, 2025.

[29] Y. Du and L. Kaelbling, "Compositional generative modeling: A single model is not all you need," *arXiv preprint arXiv:2402.01103*, 2024.

[30] Z. Yang, J. Mao, Y. Du, J. Wu, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Compositional diffusion-based continuous constraint solvers," *arXiv preprint arXiv:2309.00966*, 2023.

[31] Y. Du, S. Li, J. Tenenbaum, and I. Mordatch, "Learning iterative reasoning through energy minimization," in *International Conference on Machine Learning*, pp. 5570–5582, PMLR, 2022.

[32] S. Li, Y. Du, G. Van de Ven, and I. Mordatch, "Energy-based models for continual learning," in *Conference on lifelong learning agents*, pp. 1–22, PMLR, 2022.

[33] N. Liu, S. Li, Y. Du, A. Torralba, and J. B. Tenenbaum, "Compositional visual generation with composable diffusion models," in *European conference on computer vision*, pp. 423–439, Springer, 2022.

[34] Y. Du, T. Lin, and I. Mordatch, "Model based planning with energy based models," 2021.

[35] Y. Du, S. Li, Y. Sharma, J. Tenenbaum, and I. Mordatch, "Unsupervised learning of compositional energy concepts," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15608–15620, 2021.

[36] N. Liu, S. Li, Y. Du, J. Tenenbaum, and A. Torralba, "Learning to compose visual relations," *Advances in Neural Information Processing Systems*, vol. 34, pp. 23166–23178, 2021.

[37] Y. Du, S. Li, and I. Mordatch, "Compositional visual generation and inference with energy based models," *arXiv preprint arXiv:2004.06030*, 2020.

[38] Y. Du, S. Li, J. Tenenbaum, and I. Mordatch, "Improved contrastive divergence training of energy based models," *arXiv preprint arXiv:2012.01316*, 2020.

[39] B. Dai, Z. Liu, H. Dai, N. He, A. Gretton, L. Song, and D. Schuurmans, "Exponential family estimation via adversarial dynamics embedding," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[40] Y. Du and I. Mordatch, "Implicit generation and modeling with energy based models," *Advances in neural information processing systems*, vol. 32, 2019.

[41] Y. Du, T. Lin, and I. Mordatch, "Model based planning with energy based models," in *Conference on Robot Learning*, 2019.

[42] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 681–688, 2011.

[43] G. O. Roberts and R. L. Tweedie, "Exponential convergence of langevin distributions and their discrete approximations," *Bernoulli*, vol. 2, no. 4, pp. 341–363, 1996.

[44] R. M. Neal, "Mcmc using hamiltonian dynamics," in *Handbook of Markov Chain Monte Carlo* (S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, eds.), pp. 113–162, Chapman & Hall/CRC, 2011.

[45] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[46] R. Wang and Y. Du, "Equilibrium matching: Generative modeling with implicit energy-based models," *arXiv preprint arXiv:2510.02300*, 2025.

[47] R. Boney, J. Kannala, and A. Ilin, "Regularizing model-based planning with energy-based models," in *Conference on Robot Learning*, pp. 182–191, PMLR, 2020.

[48] T. Wu, T. Maruyama, L. Wei, T. Zhang, Y. Du, G. Iaccarino, and J. Leskovec, "Compositional generative inverse design," in *The Twelfth International Conference on Learning Representations*, 2024.

[49] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*, pp. 8748–8763, PmLR, 2021.

[50] L. P. Kaelbling and T. Lozano-Perez, "Hierarchical task and motion planning in the now," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[51] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

[52] J. Mao, T. Lozano-Perez, J. B. Tenenbaum, and L. P. Kaelbling, "Pdsketch: Integrated domain programming, learning, and planning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[53] C. Paxton, C. Xie, T. Hermans, and D. Fox, "Predicting stable configurations for semantic placement of novel objects," in *Proceedings of the Conference on Robot Learning (CoRL)*, 2022.

[54] Z. Yang, C. R. Garrett, T. Lozano-Perez, L. P. Kaelbling, and D. Fox, "Sequence-based plan feasibility prediction for efficient task and motion planning," in *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[55] Maxim Lukiyanov, Guoliang Hua, Geeta Chauhan, and Gisle Dankel, "Introducing PyTorch Profiler - the new and improved performance tool," 2021. https://pytorch.org/blog/introducing-pytorch-profiler-the-new-and-improved-performance-tool/, accessed 2021-05-21.

[56] A. Krizhevsky, V. Nair, G. Hinton, *et al.*, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, vol. 55, no. 5, p. 2, 2014.

[57] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *CVPR*, 2017.

[58] P.-W. Wang, P. Donti, B. Wilder, and Z. Kolter, "Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver," in *International Conference on Machine Learning*, pp. 6545–6554, PMLR, 2019.

[59] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[60] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[61] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pretrained transformers," *arXiv preprint arXiv:2210.17323*, 2023.

[62] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[63] Q. Liu, Y. Han, Y. Cai, T. Ge, Y. Lin, S. Liu, and V. Chandra, "Hawq-v2: Hessian aware quantization of neural networks with mixed precision," in *European Conference on Computer Vision*, 2020.

[64] T. Dettmers *et al.*, "Sparse quantized representation for large language models," *arXiv preprint arXiv:2306.03078*, 2023.

[65] B. Peng *et al.*, "H2o: Heavy-hitter oracle for efficient kv cache compression in llms," in *Advances in Neural Information Processing Systems*, 2023.

[66] T. Dao *et al.*, "Flashattention-2: Faster attention with better parallelism and work partitioning," in *International Conference on Machine Learning*, 2023.

[67] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," in *International Conference on Machine Learning*, 2021.

[68] T. Dettmers *et al.*, "Qlora: Efficient finetuning of quantized large language models," in *Advances in Neural Information Processing Systems*, 2024.

[69] J. Lin *et al.*, "Awq: Activation-aware weight quantization for llms," *arXiv preprint arXiv:2306.00978*, 2023.

[70] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[71] N. P. Jouppi *et al.*, "A domain-specific supercomputer for training deep neural networks," *Communications of the ACM*, 2020.

[72] D. Narayanan *et al.*, "Efficient large-scale language model training on gpu clusters," in *International Conference on Machine Learning*, 2021.

[73] Y. Kim *et al.*, "Spatten: Efficient sparse attention architecture for large language models," in *International Symposium on Computer Architecture (ISCA)*, 2023.

[74] S. Liu *et al.*, "Sparta: Efficient structured sparse transformer acceleration," in *MICRO*, 2023.

[75] Z. Huo, B. Gu, H. Huang, *et al.*, "Decoupled parallel backpropagation with convergence guarantee," in *International Conference on Machine Learning*, pp. 2098–2106, PMLR, 2018.

[76] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in neural information processing systems*, vol. 24, 2011.

[77] S. S. Banerjee, Z. T. Kalbarczyk, and R. K. Iyer, "Acmc 2: Accelerating markov chain monte carlo algorithms for probabilistic models," *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[78] S. Zhao, J. Yin, L. Yao, M. Andraud, W. Meert, and M. Verhelst, "Mc ̂2 a: Enabling algorithm-hardware co-design for efficient markov chain monte carlo acceleration," *arXiv preprint arXiv:2507.12935*, 2025.

[79] Y. Chai, G. G. Ko, W.-T. Mark Ting, L. Bailey, D. Brooks, and G.-Y. Wei, "Coopmc: Algorithm-architecture co-optimization for markov chain monte carlo accelerators," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 38–52, 2022.