Article:
Zou, Jie and Gray, Ian (2026) POMDP-Active Inference Model for Hybrid Critical Multi-Core Heterogeneous Scheduling. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. pp. 1211-1224. ISSN: 0278-0070

https://doi.org/10.1109/TCAD.2025.3592586

POMDP-Active Inference Model for Hybrid Critical Multi-Core Heterogeneous Scheduling

Jie Zou, Ian Gray
Department of Computer Science, University of York, UK
{*jie.zou, ian.gray*}*@york.ac.uk*

***Abstract*—In heterogeneous multi-core systems, efficiently allocating tasks to cores is a key challenge due to the complexity of hardware architectures and the dynamic nature of workloads. System states and task behaviors are inherently partially observable due to limited accessible information about interactions among tasks and shared resources, which complicates the prediction and management of execution times and resource contention. Moreover, uncertainties such as unpredictable execution times and variable workloads, combined with the challenge of prioritising high-criticality tasks while maintaining overall system performance and ensuring fairness in the execution of low-criticality tasks, necessitate the use of sophisticated allocation strategies. We propose a novel approach which embraces the uncertainty of modern systems. Partially Observable Markov Decision Processes (POMDP) and Active Inference are used to minimise uncertainty and optimise allocation decisions. By representing uncertainties within the POMDP framework, our method enables probabilistic predictions of task behaviours and system states. Active Inference refines these predictions and adapts decisions dynamically, handling workload variability and system changes such as adding new tasks. Our methodology emphasises application-layer scheduling as opposed to kernel-level modifications, thereby giving our approach both platform awareness and deployment flexibility. Moreover, the capacity for offline training mitigates the adverse effects of online updates on the accuracy of target platform comprehension. Experimental results demonstrate superior performance over baseline approaches, enhancing system efficiency and resource utilisation by incorporating task criticality awareness, even in the presence of uncertainties and partial observability.**

***Index Terms*—Partially Observable Markov Decision Processes, Active Inference, Free Energy Principle, Task Scheduling**

I. Introduction

The rise of modern high-performance System-on-Chip (SoC) platforms introduces new challenges due to the integration of heterogeneous processors, memory subsystems, and hardware accelerators. This complexity complicates task scheduling, particularly when safety standards (e.g., in automotive, aerospace, and industrial automation [1]) require criticality-aware prioritisation. Balancing stringent timing guarantees for safety-critical tasks with efficient use of shared resources adds further scheduling complexity [2].

In real-time systems, Worst-Case Execution Time (WCET) estimation has traditionally been the foundation for performance analysis. Techniques based on static analysis of control flow, hardware, and memory models [3] remain common. However, under uncertainty, these methods often produce overly conservative estimates, limiting their practicality for today's heterogeneous and dynamic systems. Increasing software complexity, frequent updates, and "black-box" components such as GPUs [4] exacerbate the difficulty of accurate modeling, making complete system knowledge—and thus precise timing prediction—unrealistic. These challenges highlight the need for adaptive scheduling strategies capable of managing uncertainty and maintaining robust performance. Uncertainties include dynamic workloads, resource contention, and fluctuating execution times, all of which affect task interactions and execution predictability. Additionally, system changes may alter both task sets and interference patterns on shared resources such as caches or memory controllers.

Ultimately, task scheduling must guarantee timely execution of safety-critical functions (e.g., brake control) without disruption from lower-criticality tasks (e.g., infotainment). This is often achieved through strict isolation via dedicated resources. However, as noted by Burns *et al.* [5], even non-safety-critical tasks can be mission-critical, requiring prioritisation. Many such tasks operate under firm or soft deadlines, and reducing deadline miss rates is key to improving system-level Quality of Service (QoS). Examples include optimal path updates in autonomous vehicles or predictive maintenance tasks—critical for system efficiency, even if occasional deadline misses are tolerable. In this work, we focus on managing shared resources among non-safety-critical tasks of varying criticality. Our goal is to ensure efficient and fair resource allocation, minimise deadline violations, and preserve overall system performance and functional integrity under uncertainty.

To address these complexities, we propose a novel framework that integrates Partially Observable Markov Decision Processes (POMDP) with Active Inference. This approach enables effective handling of uncertainties, dynamic task behaviors, and partial observability, offering an adaptive and efficient solution for resource allocation. The POMDP model frames task allocation as a decision-making process under uncertainty, dynamically updating probabilistic beliefs about system states based on incomplete observations. Actions, such as assigning tasks to specific cores, are optimised to refine these beliefs and maximise system performance. To enhance expressiveness, we replace the standard normal distribution with a multivariate Gaussian distribution, allowing the model to capture inter-variable relationships, identify sources of interference, and manage resource contention effectively. Active Inference extends the POMDP framework by introducing mechanisms for continuous learning and proactive decision-making. By leveraging Expected Free Energy (EFE), it balances two competing objectives: reducing uncertainty about system states and optimising resource allocation to meet performance goals.

Through a hybrid training approach, the framework collects data during runtime and performs offline updates, ensuring minimal computational overhead during real-time operations. This design allows the model to evolve iteratively, adapting to changing workloads and improving decision-making capabilities over time, ensuring robust performance in dynamic and uncertain environments.

The **primary contributions** of this paper are as follows:

- **Dynamic Task Handling based on Uncertainty:** The proposed framework effectively manages dynamic task behaviour, including changes in execution patterns. (e.g., periods) that alter system workload and variations in release order that affect resource contention patterns. Even when encountering unseen tasks, the framework demonstrates a notable level of controllability and outperforms traditional heuristic and machine learning approaches.
- **Criticality-Aware Scheduling:** The framework incorporates task criticality differences, prioritising higher-criticality tasks while maintaining fairness for lower-criticality tasks. This approach ensures efficient resource utilisation and equitable execution across tasks of varying importance. By dynamically adjusting allocation decisions, the framework minimises idle periods and avoids overprovisioning.
- **Platform-Agnostic:** The method is platform-agnostic, ensuring adaptability across diverse hardware configurations and workload types, making it highly scalable for future systems with increasing computational demands. Additionally, the hybrid training approach can balance computational overhead and adaptability.
- **Open-Source:** The framework includes the first open-source, end-to-end software prototype for hybrid-critical task scheduling. [1]

The rest of the paper is organised as follows: Section II introduces the background and related work. Section III details the proposed POMDP-based Active Inference method, followed by the task allocation formulation in Section IV. The evaluation is presented in Section V. Finally, Section VI concludes the paper and discusses future directions.

II. Background and Related Work

This section reviews the existing literature on task allocation strategies, handling uncertainties and partial observability, criticality-aware scheduling, interference mitigation, and adaptive learning.

Effective task allocation requires aligning tasks with the most suitable resources. Traditional heuristic-based approaches, such as Worst Fit and Best Fit, along with advanced algorithms such as HEFT and CPOP [6], [7], focus on balancing computational loads across cores. While these methods provide foundational strategies, they often neglect task-specific characteristics and the complexities introduced by dynamic workloads, leading to suboptimal performance. Additionally, the challenge is compounded when system states are not fully observable or predictable due to monitoring limitations or system complexity [4], [8]. Probabilistic models, such as those proposed in [9], [10], address uncertainties by modeling execution behaviors and system dynamics. However, standard Markov Decision Processes (MDPs) assume full state observability, limiting their applicability in multi-core systems where observability is inherently partial [11], [12].

In safety-critical systems, tasks are assigned high-criticality levels to meet stringent timing constraints even under uncertainty [13], [14], [15]. Mixed-criticality scheduling algorithms, such as those in [16], [17], rely on Worst-Case Execution Time (WCET) estimates to guarantee deadlines for high-criticality tasks. However, accurately estimating WCET in complex, partially observable systems remains a challenge. Resource contention and interference further complicate task execution, as shared system resources (e.g., caches, memory bandwidth) introduce variability. Techniques like task isolation, priority-based scheduling, and temporal partitioning [18], [19] mitigate interference but often underutilise system resources and struggle to scale with increasing system complexity.

Advanced interference mitigation strategies include cache partitioning [20], [21], [3], temporal partitioning frameworks [22], and dynamic memory bandwidth allocation [23]. Recent models, such as the Inter-Task Interference Matrix (ITIM) [24], quantify cache and memory interference to enhance scheduling efficiency. However, platform-specific approaches relying on hardware features like Intel's Cache Allocation Technology (CAT) limit generalisability across heterogeneous architectures. These solutions often fail to address the broader interdependencies between tasks and resources in diverse or evolving multi-core environments.

Machine learning (ML) techniques have advanced task scheduling, offering predictive and adaptive capabilities for real-time systems. For example, statistical and deep learning models [25] predict task interference on Commercial Off-The-Shelf (COTS) platforms but often operate as "black boxes", and thereby limiting the interpretability which is essential for safety-critical systems. Other approaches, such as cache-recency-based task allocation [26], improve resource reuse but rely heavily on empirical data and lack scalability across heterogeneous cores. Reinforcement learning (RL) frameworks [27], [28] optimise resource allocation dynamically but often focus on isolated resources (e.g., CPU, memory) without fully addressing inter-core interference. They do not explicitly address timing guarantees or real-time scheduling overheads. The authors of [29], [30] employed recurrent neural networks (RNNs) to enhance system performance. However, as deep learning models, RNNs often suffer from a lack of explainability, making them less suitable for applications that require transparency in decision-making processes. Moreover, while these works demonstrate the use of RNNs for performance improvement, they do not adequately address whether the trained agents can effectively handle workload dynamics. As systems scale to many-core, and heterogeneous architectures, traditional and ML-based approaches face challenges in holistically managing task dependencies, resource interdependencies, and the complexity of real-time constraints. Narrow optimisation focuses often result in suboptimal resource allocation, leaving gaps in performance when addressing broader system dynam-

[1]The link to the code is omitted for review and will be made publicly available in the final version.

ics [31].

POMDP extend MDPs to address decision-making under partial observability [32], [33]. POMDPs have been applied in fields such as robotics and autonomous navigation [34], [35], providing a principled framework to model uncertainties and partial observability. Despite their potential, the application of POMDPs to task allocation in heterogeneous multi-core systems remains underexplored. The computational complexity of solving POMDPs grows exponentially with state space size [36], necessitating efficient state-space design and interference modeling techniques, as explored in [37].

Active Inference, rooted in neuroscience, minimises Expected Free Energy (EFE) to balance prediction accuracy and belief complexity [38], [39]. It unifies perception, learning, and action into a single process, naturally integrating uncertainty and partial observability. In task allocation, Active Inference enables systems to learn and adapt continuously, refining models based on new observations. This dynamic adaptability is essential for handling evolving workloads, unforeseen task behaviors, and changes in system topology. Integrating Active Inference into task allocation introduces challenges, such as computational overhead and model stability. Balancing update frequency and granularity ensures efficient belief updates without disrupting real-time performance. By incorporating adaptive update mechanisms, Active Inference supports robust and scalable decision-making in dynamic, heterogeneous environments. While its application in computing systems remains emerging, the potential benefits make it a promising avenue for future research.

III. Method Overview

In this section we propose a novel approach that leverages POMDP combined with Active Inference to solve the task-to-core allocation problem.

A. Design Intentions and Architecture

We frame the task allocation problem as a POMDP. Recognising the exponential growth in monitoring overhead and computational complexity with increasing state space size we focus exclusively on a subset of key performance indicators (KPIs). These KPIs are identified through workload profiling and interference analysis for each specific case. For instance, a speed-critical system might prioritise execution latency and response time, while a low-power system might emphasise energy efficiency and thermal management. Our framework is KPI-agnostic, allowing it to adapt to different use cases by considering the relevant KPIs for each scenario. While the observed KPIs guide the model, some system parameters and performance metrics remain unobservable or difficult to monitor directly. These unobserved elements are treated as latent uncertainties within the POMDP framework, enabling the model to account for incomplete information. By incorporating both observable and latent factors, the framework remains robust and adaptable, effectively balancing the trade-offs between computational overhead and system performance across diverse application domains. Figure 1 presents the architecture of the proposed POMDP-based Active Inference

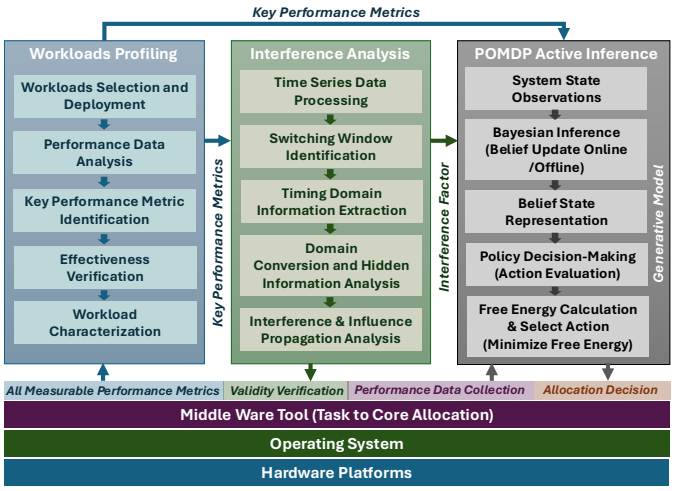

Fig. 1. The proposed architecture.

methodology. To maintain conciseness, detailed methodologies for workload profiling and interference analysis are covered in a separate paper, and this section provides a high-level overview. This work focuses on application-layer scheduling at the middleware level (Figure 1). To meet the needs of safety-critical systems, we adopt a static partitioning paradigm, where task-to-core mappings are determined at initialisation or during coordinated reconfiguration events triggered by system dynamics, such as changes in task release order or execution period. Tasks are pinned to cores, executed periodically and non-preemptively, and do not migrate at runtime. Our method does not modify OS-level schedulers. Platform-induced uncertainties are captured in training KPIs, making the approach platform-aware and deployment-flexible, suitable for user-space or future kernel-level integration. The focus on the application layer aligns with the "shielding" concept proposed by Alshiekh et al. [40] for reinforcement learning in safety-critical systems—an approach Shi et al. [41] later discussed for ensuring both functionality and timeliness in real-time systems. However, addressing timing guarantees or the overheads associated with real-time scheduling remains an open question.

Workloads profiling: We first collect performance counters during isolated execution of each task. By ensuring no interference from competing tasks, the collected data serves as a clean baseline for analysis. This employs machine learning classifiers, such as Random Forest (RF), for feature selection, emphasising simplicity, explainability, and effectiveness with limited data. The use of RF-based classification enables the identification of the most critical features, facilitating efficient KPI selection while minimising the number of required monitoring parameters. This approach is particularly beneficial in minimising the POMDP model's computational overhead as it ensures that only the most informative features are retained for decision-making.

Interference analysis: We then evaluate the impact of task interference by executing tasks concurrently, both within the same category and across different categories. Relationships between various contention sources are modeled using a graph,

where nodes represent sources of contention and monitored factors, while edges capture the interactions and influence propagation between these sources. To analyse the graph, the Clauset-Newman-Moore (CNM) modularity maximisation technique is applied to identify communities within the graph structure. Particular attention is given to key communities that significantly contribute to increased execution times. We use an interference factor to quantify the relationship between variations in execution time and fluctuations in key contention resources within these critical communities. Then, we integrate the factor into the observation model of our POMDP framework to empower the model to account for dynamic contention effects during decision-making.

Scalability through Hierarchical Partitioning and Local Inference: In high-integrity systems, scalability must be achieved through architectural decomposition. As outlined in standards like ISO 26262-6:2018 [42] and AUTOSAR Adaptive Platform R23-11 [43], timing and schedulability guarantees should be validated per software unit or execution cluster, enabling modular verification and compositional safety analysis. Our approach follows this principle by employing a hierarchical architecture, where functional requirements are mapped onto hardware resources in layers. Instead of a monolithic scheduler, we deploy POMDP-based Active Inference models at the cluster level. Each model manages a smaller, localised task set, managing the complexity of state and observation spaces while maintaining adaptability under uncertainty. This composable design supports incremental adoption across heterogeneous multicore systems. While a full system-level scalability analysis is beyond this paper's scope, the presented framework is structured to support such extensions. This work focuses on evaluating the intra-cluster inference mechanism within a defined, safety-compliant context, that prioritises local predictability and partitioned validation as foundations for scalable system design [44].

B. POMDP Active Inference Model

Figure 2 illustrates the workflow of our proposed POMDP Active Inference Model, structured as a continuous cycle of perception, prediction, action, and adaptation to support ongoing optimisation - and optionally ongoing learning - in dynamic environments. The workflow begins with defining the model's core components, including the state space, action space, initial belief, transition function, and observation model. Depending on deployment requirements, users can select between two training modes: online updating for real-time adaptation or offline training to reduce computational overhead and the impact of model updates. In the online mode, the system continuously monitors key performance metrics and updates the model in real time, ensuring immediate adaptability to changes. In offline training, tasks are executed in dry runs to collect data and train the model without impacting the primary system, offering flexibility for operational requirements.

Task allocation is treated as an action within the model. During the training phase, tasks are randomly assigned to the cores, and the belief state is updated based on observed performance metrics for each action. After training, parameter

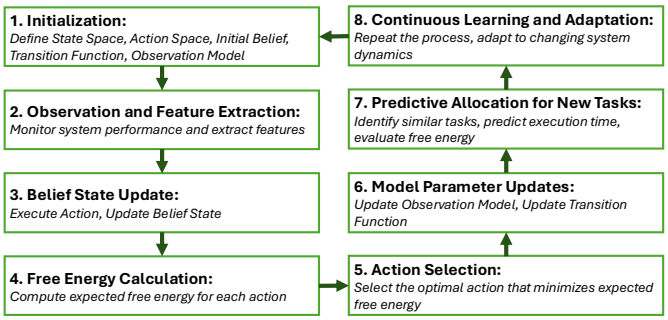

Fig. 2. The workflow of POMDP Active Inference

matrices are fixed to facilitate Expected Free Energy (EFE) calculations. At runtime, each potential allocation action is evaluated using EFE to balances exploration (uncertainty reduction) and exploitation (leveraging known information), to minimise decision uncertainty, and to optimise task-to-core allocations. For users opting for online updates, the observation model and transition function are continuously refined with new data, enhancing predictions and adapting to real-time system changes. In offline training, while parameters remain fixed during operation, the system collects data for periodic updates.

A key strength of this approach lies in its ability to handle previously unseen tasks. When encountering a new task, the model leverages key performance metrics identified during workload profiling to locate similar tasks within the training dataset. Using distance metrics such as Euclidean distance, the model estimates execution times and calculates EFE to enable efficient resource allocation for unfamiliar tasks. In essence, the model can evaluate what "kind" a new task is and schedule it accordingly without explicit knowledge of the hardware or software. Whether updated online or offline, as demonstrated in Section V, this approach outperforms other methods by maintaining superior task allocation efficiency and adaptability, even in scenarios with high variability and unseen workloads. These results underscore the robustness of the proposed methodology in handling diverse and unpredictable operational contexts.

C. Key Concepts

This section introduces the key components of our approach, before later sections which expand on these concepts in mathematical detail.

*1) **Partially Observable Markov Decision Processes**:* POMDP is an advanced framework for decision-making in situations where the system's state cannot be fully observed, making it highly suitable for real-world applications with uncertain environments. Key components of a POMDP include:

- **State Space**: All possible conditions or configurations of the system. In this work, each state represents a complete set of task-to-core assignments, and the effect of those assignments.
- **Action Space**: Actions assign a task to a core, thereby moving from one state to another.

- **Observations**: Due to partial observability, the model relies on observable data to infer the system's state. In our model, observations are derived from key performance metrics identified through workload profiling and interference analysis, and provide indirect evidence about the true state, helping refine the model's beliefs over time.
- **Belief State**: A probabilistic distribution over possible states, reflecting the agent's current estimate of the system's condition based on available observations. It is continuously updated as new data is acquired, enabling the model to learn and improve its understanding of the system dynamics and allocation effects.
- **Transition Function**: Defines the probability of moving from one state to another after an action is taken. Models how the system evolves in response to task allocations and captures dependencies between tasks.
- **Observation Model**: The probability of receiving a specific observation given the system's underlying state. It helps determine how likely each observation is under various states, providing a mechanism to compare the observed outcomes to expected outcomes and refine the belief state.
- **Free Energy Minimisation**: Traditionally, a POMDP includes a reward function that assigns a value to each action-state combination, guiding the decision-making toward maximising cumulative rewards. In the Active Inference framework, this is replaced by free energy minimisation, where the goal is to choose actions that minimise uncertainty (free energy).

These components support decision-making in uncertain environments. By updating beliefs, evaluating transition probabilities, and selecting actions that minimise EFE, a POMDP model can effectively handle the complexity and variability of task-to-core allocation in multi-core systems.

*2) **Multivariate Gaussian Distribution**:* The Multivariate Gaussian Distribution is a widely adopted method in probabilistic modeling for representing high-dimensional data with interdependent variables [45]. In our POMDP Active Inference framework, we move beyond the original assumption of independent, normally distributed observations by adopting a multivariate Gaussian, allowing us to capture both the variability of individual performance features (such as execution time) and their interdependencies (e.g., between cache misses and instruction counts). This interconnected view is critical for precise task-to-core allocation, as it helps the model understand how fluctuations in one metric influence others. Furthermore, the multivariate Gaussian enables efficient closed-form Bayesian updates for belief state refinement and EFE computation, which is crucial for maintaining low computational overhead in real-time systems.

Real-world execution behaviours often deviate from strict normality; however, our goal is not perfect density estimation, but practical and effective decision-making under uncertainty. In our case study, we observed that task performance metrics — when aggregated under our selected profiling scenarios — approximated unimodal distributions, which supports the pragmatic use of a Gaussian model in this context. Adopting more complex models, such as mixture models or non-parametric approaches, would significantly increase computational complexity, potentially exceeding real-time constraints. Our approach strikes a balance between expressiveness and efficiency, while retaining extensibility — the framework can readily accommodate richer models in future work, if necessary, without fundamental architectural changes.

*3) **The Free Energy Principle**:* The Free Energy Principle (FEP), rooted in Bayesian inference, is a decision-making framework which aims to reduce uncertainty. At its core, free energy minimisation seeks to align model predictions (e.g., expected execution times, resource conflicts, and performance requirements) with actual observed outcomes, thereby reducing the "surprise" or deviation between them. In the Active Inference implementation of this work, as new observations are received the model iteratively updates its belief states and refines its action policies, recalculating EFE to optimise decision-making. This process ensures that allocation decisions adapt to evolving system conditions and align with performance objectives.

IV. Framework Formulation

In this section, we present the core formulation of our POMDP-based Active Inference model, focusing on its key components and decision-making process. Using illustrative examples, we explain how the model processes inputs and derives task-to-core allocation decisions under uncertainty.

A. POMDP Framework

The POMDP framework models the uncertainty in system dynamics. By representing the task-to-core allocation as a POMDP, we use probabilistic methods to make decisions, considering both current knowledge and future uncertainties. POMDP is defined by the tuple $(S, A, \mathbf{o}, \mathcal{T}, \mathcal{O}, R)$, where S represents a set of states, i.e., possible task-to-core allocations. Each state $s \in S$ represent a possible assignment of task τ to core c. $s = (\tau_i, c_j), i \in \{1, 2, \cdots, N_\tau\}, j \in \{1, 2, \cdots, N_c\}$, where N_τ is the number of tasks and N_c is the number of cores. The state space includes all combinations of tasks and cores, representing all possible allocation scenarios. A is a set of actions corresponding to allocation decisions. Each action $a \in A$ corresponds to the decision to assign task τ to core c, $a = (t, c)$. The action space mirrors the state space, representing the set of possible allocation decisions. Observations ($\mathbf{o}$) are vectors of system performance metrics and features observed after allocating a task to a core $\mathbf{o} = [o_1, o_2, \cdots, o_k] \in \mathbb{R}^K$, where K is the number of observation features. $\mathcal{T}(s'|s, a)$ represents the state transition probability function. $O(\mathbf{o}|s', a)$ is the observation probability function. $R(s, a)$ is the reward function in conventional POMDP; in our model, it is represented via free energy minimisation. Here, we provide a set of example components used in our analysis. These components include: *Execution Time, Instructions, Branch Misses, Period, Ideal Execution Time, Interference Factor, Balance, Core Speed, Task Criticality (Numeric), Deadline Missed (Binary), Increased Instructions, Increased Branch Misses, Cache References, Instructions F1F2DV, Cache References F1F2DV, and Branch Misses F1F2DV*. These components enable the model

to make context-aware allocation decisions. The F1F2DV metrics are frequency domain (harmonic) changes in response to interference.

B. Transition Function $\mathcal{T}$

The transition function $\mathcal{T}(s'|s,a)$ defines the probability of transitioning to state s' from state s after taking action a following Equation (1). It models the dynamics of task allocation under uncertainty.

$$\mathcal{T}(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a) \tag{1}$$

If prior information is available, we can initialise the transition function $\mathcal{T}$ accordingly - i.e. from a previously-trained model. If not then we can assign equal probabilities to all possible transitions from a given state-action pair, $\mathcal{T}(s'|s,a) = 1/|S|$, or initialise the transition probabilities using a Dirichlet prior to ensure that probabilities are valid and sum to one, $\mathcal{T}(s'|s,a) \sim Dirichlet(\alpha)$, where α is a hyperparameter vector representing prior counts. This ensures that the transition probabilities are valid (non-negative and summing to one), provides a probabilistic foundation for Bayesian updates, and can incorporate prior knowledge about likely transitions by adjusting α. This approach is particularly beneficial in handling sparse data, avoiding zero probabilities for rarely observed transitions, and enabling robust learning even with limited initial data. Additionally, it regularises the learning process, prevents overfitting to noisy observations, and facilitates smoother convergence. The Dirichlet prior is our preferred approach.

As the system operates, we update $\mathcal{T}$ according to Equation (2) based on observed transitions using transition counts $N(s,a,s')$, which represents the number of times we observed a transition from state s to state s' after action a. The denominator represents the total number of transitions observed from state s after action a.

$$\mathcal{T}(s'|s,a) = \frac{N(s,a,s')}{\sum_{s'' \in S} N(s,a,s'')} \tag{2}$$

When a new transition is observed, we need to increment the count following $N(s,a,s') \leftarrow N(s,a,s') + 1$ and then recalculate the probabilities for all s' according to Equation (2). Please note the transition probabilities are governed by the laws of probability and must satisfy $\mathcal{T}(s'|s,a) \geq 0 \ \forall s, s', a$ and $\sum_{s' \in S} \mathcal{T}(s'|s,a) = 1 \ \forall s, a$.

Example: Consider a simplified system with two tasks (τ_1, τ_2) and two cores (c_1, c_2). The states $\mathcal{S} = \{(\tau_1, c_1), (\tau_1, c_2), (\tau_2, c_1), (\tau_2, c_2)\}$ and the action space is the same as states. Suppose we have observed the following transitions after taking action $a = (\tau_1, c_1)$: (1) 3 times from state $s = (\tau_1, c_1)$ to $s' = (\tau_2, c_1)$ and (2) 1 time from state $s = (\tau_1, c_1)$ to $s' = (\tau_1, c_2)$. The transition counts are: $N((\tau_1, c_1), (\tau_1, c_1), (\tau_2, c_1)) = 3$ and $N((\tau_1, c_1), (\tau_1, c_1), (\tau_1, c_2)) = 1$. Transition probabilities are then: $\mathcal{T}((\tau_2, c_1)|(\tau_1, c_1), (\tau_1, c_1)) = 3/(3+1) = 0.75$ and $\mathcal{T}((\tau_1, c_2)|(\tau_1, c_1), (\tau_1, c_1)) = 1/(3+1) = 0.25$. This indicates that, after allocating τ_1 to c_1, there is a 75% chance that the system will transition to a state where τ_2 is allocated to c_1 and a 25% chance of transitioning to a state where τ_1 is moved to c_2.

C. Observation Model O

The observation model $O(\mathbf{o}|s',a)$ captures the probability of observing $\mathbf{o}$ when the system is in state s' after action a, as shown in Equation (3). We assume that observations are drawn from a multivariate Gaussian distribution, with observation probabilities influenced by both the current state and the preceding action. Thus, the observation model can be formally expressed as shown in Equation (4).

$$O(\mathbf{o}|s',a) = P(\mathbf{o}_{t+1} = \mathbf{o}|s_{t+1} = s', a_t = a) \tag{3}$$

$$O(\mathbf{o}|s',a) = \mathcal{N}(\mathbf{o}; \mu_{s',a}, \mathbf{\Sigma}_{s',a}) \tag{4}$$

Where $\mu_{s',a}$ is a mean vector of observations for state s' and action a; $\mathbf{\Sigma}_{s',a}$ represents the covariance matrix representing the variability of observations. This Gaussian assumption allows us to model the continuous and potentially correlated nature of performance metrics.

The observation model is essential for both belief state update and free energy computation. The model integrates new observations to refine the model's estimate of the system's true state. For free energy computation, the model quantifies the "surprise" or discrepancy between predicted and actual observations, which directly impacts the model's action selection by encouraging choices that reduce this surprise. At the beginning of the process, we initialise the observation model parameters for each state-action pair. We initialise $\mu_{s',a}$ to zeros and $\mathbf{\Sigma}_{s',a}$ as diagonal matrices with small positive values on the diagonal (e.g., $0.1 \times \mathbf{I}$), where $\mathbf{I}$ is the identity matrix. This initialisation reflects our initial uncertainty about the observations associated with each state-action pair.

As observations are collected during the operation of the system, we update the observation model parameters to better reflect the observed data. The mean vector $\mu_{s',a}$ is updated incrementally using a learning rate α following Equation (5).

$$\mu_{s',a} \leftarrow \mu_{s',a} + \alpha(\mathbf{o} - \mu_{s',a}) \tag{5}$$

$\mathbf{o}$ is the new observation vector and α is the learning rate ($0 < \alpha < 1$). This update rule moves the mean towards the new observation, with the rate of adjustment controlled by α. The covariance matrix $\mathbf{\Sigma}_{s',a}$ is updated to capture the variability in the observations following Equation (6).

$$\mathbf{\Sigma}_{s',a} \leftarrow (1-\alpha)\mathbf{\Sigma}_{s',a} + \alpha(\mathbf{o} - \mu_{s',a})(\mathbf{o} - \mu_{s',a})^\top \tag{6}$$

This update rule adjusts the covariance matrix based on the deviation of the new observation from the updated mean. To ensure numerical stability and that $\mathbf{\Sigma}_{s',a}$ remains positive definite, we apply regularisation techniques, including eigenvalue clipping and shrinkage. First of all, we compute the eigenvalues λ_i and eigenvectors $\mathbf{v}_i$ of $\mathbf{\Sigma}_{s',a}$. Then, set $\lambda_i' = \max(\lambda_i, \epsilon)$, where $\epsilon > 0$ is a small threshold. After that, we can reconstruct the covariance matrix as:

$$\mathbf{\Sigma}_{s',a} = \sum_i \lambda_i' \mathbf{v}_i \mathbf{v}_i^\top \tag{7}$$

Then, we apply a shrinkage blend to $\mathbf{\Sigma}_{s',a}$ by combining it with a scaled identity matrix, which can be expressed as:

$$\mathbf{\Sigma}_{s',a} \leftarrow (1-\lambda)\mathbf{\Sigma}_{s',a} + \lambda\mathbf{I} \tag{8}$$

where λ is the shrinkage parameter, controlling the balance between the original covariance $\mathbf{\Sigma}_{s',a}$ and the scaled identity matrix $\mathbf{I}$. This adjustment helps ensure numerical stability and robustness, especially in cases where $\mathbf{\Sigma}_{s',a}$ may be poorly conditioned or singular, which is crucial for computing the multivariate Gaussian likelihood. Then, the likelihood of an observation given a state and action is computed as:

$$O(\mathbf{o}|s',a) = \mathcal{N}(\mathbf{o};\mu_{s',a},\mathbf{\Sigma}_{s',a}) = \frac{1}{\sqrt{(2\pi)^k|\mathbf{\Sigma}_{s',a}|}}\exp\left(-\frac{1}{2}(\mathbf{o}-\mu_{s',a})^\top\mathbf{\Sigma}_{s',a}^{-1}(\mathbf{o}-\mu_{s',a})\right) \tag{9}$$

Where $|\mathbf{\Sigma}_{s',a}|$ is the determinant of the covariance matrix. $\mathbf{\Sigma}_{s',a}^{-1}$ represent the inverse of the covariance matrix. k is the number of observation components.

D. Belief State Update

In a POMDP, the model does not have direct access to the true state s but maintains a belief state $b(s)$, a probability distribution over all possible states. The belief state is crucial as it captures the uncertainty inherent in partially observable environments, providing a probabilistic representation of the model's current understanding of the system state. Actions are selected based on the belief state to optimise expected outcomes under uncertainty. Furthermore, the belief state is continually updated with new observations, enabling the model to adapt dynamically to changes in the environment and improve decision-making over time.

The belief state is updated in two main steps - prediction and update. In the prediction step, we estimate the next belief state $b'(s')$ based on the current belief state $b(s)$ and the action a, following Equation (10). This equation incorporates all possible state transitions from the current belief state under action a. Then, when moving to the update stage, we incorporate the new observation $\mathbf{o}$ to refine the belief state following Equation (11). The denominator ensures that the updated belief state is a valid probability distribution (sums to 1).

$$b'(s') = \sum_{s\in S} b(s)T(s'|s,a) \tag{10}$$

$$b(s') = \frac{O(\mathbf{o}|s',a)b'(s')}{\sum_{s'\in S}O(\mathbf{o}|s',s)b'(s')} \tag{11}$$

Computing products of probabilities can lead to numerical underflow when dealing with small probability values. To mitigate this, we perform computations in the logarithmic domain.

Example: Consider a simplified system with three states s_1, s_2, s_3 and current belief state is:

$$b(s) = \begin{cases} 0.5 & \text{if} \quad s = s_1 \\ 0.3 & \text{if} \quad s = s_2 \\ 0.2 & \text{if} \quad s = s_3 \end{cases} \tag{12}$$

When taking action taken a, the transition probabilities are:

$$\mathcal{T}(s'|s,a) = \begin{cases} 0.7 & \text{if} \quad s' = s_1 \ \& \ s = s_1 \\ 0.3 & \text{if} \quad s' = s_2 \ \& \ s = s_1 \\ 0.2 & \text{if} \quad s' = s_3 \ \& \ s = s_1 \\ \dots & \text{Similarly for other } s \text{ and } s' \end{cases} \tag{13}$$

The observation likelihoods are:

$$O(\mathbf{o}|s',a) = \begin{cases} 0.9 & \text{if} \quad s = s_1 \\ 0.6 & \text{if} \quad s = s_2 \\ 0.2 & \text{if} \quad s = s_3 \end{cases} \tag{14}$$

Following Equation(10), we can compute $b'(s')$ for each s'. For $s' = s_1$:

$$\begin{aligned} b'(s_1) &= \sum_{s\in S} b(s)T(s_1|s,a) \\ &= \mathcal{T}(s_1|s_1,a)b(s_1) + \mathcal{T}(s_1|s_2,a)b(s_2) + \mathcal{T}(s_1|s_3,a)b(s_3) \\ &= 0.7\times 0.5 + 0.3\times 0.3 + 0.2\times 0.2 \\ &= 0.35 + 0.09 + 0.04 = 0.48 \end{aligned} \tag{15}$$

Similarly, we can get the value for $b'(s_2)$ and $b'(s_3)$ and compute the unnormalised updated belief (the numerator of Equation (11)). For $s' = s_1$:

$$b'(s_1) = O(\mathbf{o}|s_1,a)b'(s_1) = 0.9\times 0.48 = 0.432 \tag{16}$$

Similarly, we can compute $b'(s_2)$ and $b'(s_3)$ and suppose $b'(s_2) = 0.144$ and $b'(s_3) = 0.032$. Then we can get the normalised updated belief:

$$\begin{aligned} b(s_1) &= \frac{b'(s_1)}{b'(s_1)+b'(s_2)+b'(s_3)} \\ &= \frac{0.432}{0.432+0.144+0.032} \approx 0.7118 \end{aligned} \tag{17}$$

Similarly, we can find $b(s_2) \approx 0.2368$ and $b(s_3) \approx 0.0526$. The updated belief state is:

$$b(s) = \begin{cases} 0.7118 & \text{if} \quad s = s_1 \\ 0.2368 & \text{if} \quad s = s_2 \\ 0.0526 & \text{if} \quad s = s_3 \end{cases} \tag{18}$$

Compared with the previous belief state, as shown in the Equation (12), the probability mass has shifted towards s_1, indicating increased confidence that the system is in state s_1 after incorporating the new observation. The updated belief state integrates both the predicted transitions and the latest observation, yielding a refined estimate of the system state. This updated belief state enables the model to assess potential actions by calculating EFE, guiding action selection towards optimal outcomes. Moreover, updating the belief state allows the model to adapt to unforeseen changes in the environment. By shifting probability mass toward states that better account for recent observations, the model adjusts dynamically, enhancing its responsiveness to unexpected variations and improving decision-making accuracy.

E. Free Energy Minimisation and Action Selection

Active Inference leverages the concept of EFE to make decisions that minimise uncertainty and expected surprise in future observations. The free energy F quantifies the difference between the predicted model and the observed data, effectively balancing exploration and exploitation. The EFE $F(a)$ for taking action a is defined as:

$$F(a) = \mathbb{E}_{b(s')}[\underbrace{D_{\text{KL}}\left[b(\mathbf{o} \mid s') \parallel P(\mathbf{o} \mid s', a)\right]}_{\text{Expected Surprise}} + \underbrace{D_{\text{KL}}\left[b(s') \parallel P(s' \mid s, a)\right]}_{\text{Expected Divergence}}] \tag{19}$$

Where $b(s')$ is the predicted belief state after taking action a. $b(\mathbf{o}|s')$ is the predicted observation distribution under the belief state $b(s')$. $P(\mathbf{o}|s', a)$ is the likelihood of observing $\mathbf{o}$ given state s' and action a. $P(s'|s, a)$ is the transition probability from state s to s' given action a. D_{KL} is the Kullback-Leibler (KL) divergence. In our model, the KL divergence is used to quantify the "distance" between the model's beliefs and the true probability distributions governing observations and state transitions.

Computing the KL divergence terms directly can be computationally intensive, especially in high-dimensional spaces or when the probability distributions are complex. To make the computation tractable, we aim to simplify $F(a)$ by breaking it down into components that are easier to compute. Besides, each component has a clear interpretation, facilitating an understanding of how different factors contribute to the model's decision-making process. The KL divergence between two distributions $Q(s)$ and $P(x)$ is defined as: $D_{KL}[Q(x)\|P(x)] = \sum_x Q(x) \ln Q(x)/P(x)$. Applying this in $F(a)$:

$$D_{\text{KL}}\left[b(\mathbf{o} \mid s') \parallel P(\mathbf{o} \mid s', a)\right] = \sum_o b(o|s') \ln \frac{b(o|s')}{P(o|s', a)} \tag{20}$$

$$D_{\text{KL}}\left[b(s') \parallel P(s'|s, a)\right] = \sum_{s'} b(s') \ln \frac{b(s')}{P(s'|s, a)} \tag{21}$$

The EFE becomes:

$$F(a) = \sum_{s'} b(s') \left[\sum_o b(o|s') \ln \frac{b(o|s')}{P(o|s', a)} + \ln \frac{b(s')}{P(s'|s, a)} \right] \tag{22}$$

Regarding the expected KL divergence between predicted and actual observations, we have the following:

$$\begin{aligned} &\mathbb{E}_{b(s')}[D_{\text{KL}}\left[b(\mathbf{o} \mid s') \parallel P(\mathbf{o} \mid s', a)\right]] \\ &= \sum_{s'} b(s') \left[\sum_o b(o|s') \ln \frac{b(o|s')}{P(o|s', a)} \right] \\ &= \sum_{s'} b(s') \underbrace{\sum_o b(o|s') \ln b(o|s')}_{-H(b(o,s'))} - \\ &\qquad \underbrace{\sum_{s'} b(s') \sum_o b(o|s') \ln P(o|s', a)}_{-\mathbb{E}_{b(s')}[\mathbb{E}_{b(o|s')}[\ln P(o|s', a)]]} \end{aligned} \tag{23}$$

$-H(b(o, s'))$ represents the negative entropy of the predicted observations, and the second term is the expected negative log-likelihood of observations, which captures how surprising the observations are expected to be, given action a.

When moving to the expected KL divergence between predicted and actual state transitions:

$$\begin{aligned} &\mathbb{E}_{b(s')}[D_{\text{KL}}\left[b(s') \parallel P(s'|s, a)\right]] = \sum_{s'} b(s') \ln \frac{b(s')}{P(s'|s, a)} \\ &= \sum_{s'} b(s')[\ln b(s') - \ln P(s'|s, a)] \\ &= \underbrace{\sum_{s'} b(s') \ln b(s')}_{-H(b(s'))} - \underbrace{\sum_{s'} b(s') \ln P(s'|s, a)}_{-\mathbb{E}_{b(s')}[\ln P(s'|s,a)]} \end{aligned} \tag{24}$$

$-H(b(s'))$ is the negative entropy of the predicted belief state, and the second term represents the expected negative log-transition probability, which encourages the agent to prefer actions that lead to predictable state transitions. Then, in the general case, the EFE can be written as:

$$\begin{aligned} F(a) = &-\mathbb{E}_{b(s')}[\mathbb{E}_{b(o|s')}[\ln P(o|s', a)]] - \mathbb{E}_{b(s')}[\ln P(s'|s, a)] \\ &- \sum_{s'} b(s') H(b(o, s')) - H(b(s')) \end{aligned} \tag{25}$$

Entropy (i.e., $-H(b(o, s')), -H(b(s'))$) is a measure of uncertainty associated with a probability distribution. Including the entropy term in the free energy encourages the model to consider actions that reduce uncertainty in its belief state. By minimising entropy, the agent prefers actions that lead to more certain outcomes.

In this work, we assume that the agent's beliefs about observations match the likelihood function, i.e., $b(\mathbf{o}|s') = P(\mathbf{o}|s', a)$ and $\mathbb{E}_{b(s')}[D_{\text{KL}}\left[b(\mathbf{o} \mid s') \parallel P(\mathbf{o} \mid s', a)\right]] = 0$. There for Equation (25) can be simplified to:

$$F(a) = -\mathbb{E}_{b(s')}[\ln P(s'|s, a)] - H(b(s')) \tag{26}$$

However, even with this assumption, the observations themselves may still carry uncertainty. It's important to include the expected negative log-likelihood of observations in the free energy computation to capture the inherent uncertainty and expected "surprise" associated with observations, which is crucial for the agent's decision-making process. Even though the entropy term and the expected negative log-likelihood term cancel out, the expected negative log-likelihood of observations can still be considered separately in the context of the agent's EFE. Therefore, we include $-\mathbb{E}_{b(s')}[\ln P(o|s', a)]$ separately, which represents the expected surprise or uncertainty associated with observations given the predicted states and action a. In this way, even if the agent's belief matches the observation model, there is inherent uncertainty or variability in the observations that affect decision-making. Besides, the model may prefer actions that lead to less surprising (more predictable) observations, which is captured by minimising this term. Then, the simplified EFE can be re-written as:

$$\begin{aligned} F(a) = &-\mathbb{E}_{b(s')}[\ln P(o|s', a)] - \mathbb{E}_{b(s')}[\ln P(s'|s, a)] \\ &- H(b(s'))) \end{aligned} \tag{27}$$

At each decision point, the agent computes the EFE for all possible actions according to Equation (27) and selects the one a^* with the lowest free energy:

$$a^* = \arg\min_{a \in \mathcal{A}} F(a) \tag{28}$$

To make the computation tractable, we assume Markovian transitions: that is, each transition depends solely on the current state and action. Additionally, we assume conditional independence of observations, meaning that given the current state and action, the next observation is independent of previous states and observations.

Example of Free Energy Computation: Consider a simplified scenario with two possible states s_1, s_2 and one action a. We have Belief state $b(s_1) = 0.6$, $b(s_1) = 0.4$, transition probabilities $\mathcal{T}(s_1|s_1, a) = 0.8$, $\mathcal{T}(s_2|s_1, a) = 0.2$, $\mathcal{T}(s_1|s_2, a) = 0.3$, $\mathcal{T}(s_2|s_2, a) = 0.7$, and observation likelihoods $O(\mathbf{o}_1|s_1, a) = 0.9$, $O(\mathbf{o}_2|s_1, a) = 0.1$, $O(\mathbf{o}_1|s_2, a) = 0.4$, $O(\mathbf{o}_2|s_2, a) = 0.6$. We can compute the predicted belief state b(s'):

$$\begin{aligned} b(s_1) &= \sum_s \mathcal{T}(s_1|s, a)b(s) \\ &= \mathcal{T}(s_1|s_1, a)b(s_1) + \mathcal{T}(s_1|s_2, a)b(s_2) \\ &= 0.8 \times 0.6 + 0.3 \times 0.4 = 0.6 \end{aligned} \tag{29}$$

Then. we have $b(s_2) = 1 - b(s_1) = 0.4$. Assuming we observe $\mathbf{o} = \mathbf{o}_1$, the expected negative log-likelihood can be computed as follows:

$$\begin{aligned} &- \mathbb{E}_{b(s')}[\ln P(\mathbf{o}|s', a)] \\ &= -[b(s_1) \ln P(\mathbf{o}_1|s_1, a) + b(s_2) \ln P(\mathbf{o}_1|s_2, a)] \\ &= -[0.6 \times \ln 0.9 + 0.4 \times \ln 0.4] \\ &= -[0.6 \times -0.1054 + 0.4 \times -0.9163] = 0.4297 \end{aligned} \tag{30}$$

The expected negative log-transition probability can be calculated as follows:

$$\begin{aligned} &- \mathbb{E}_{b(s')}[\ln \mathcal{T}(s'|s', a)] \\ &= -[b(s_1)(\mathcal{T}(s_1|s_1, a) \ln \mathcal{T}(s_1|s_1, a) \\ &+ \mathcal{T}(s_2|s_1, a) \ln \mathcal{T}(s_2|s_1, a)) \\ &+ b(s_2)(\mathcal{T}(s_1|s_2, a) \ln \mathcal{T}(s_1|s_2, a) \\ &+ \mathcal{T}(s_2|s_2, a) \ln \mathcal{T}(s_2|s_2, a))] \\ &= -[0.6 \times (0.8 \times \ln 0.8 + 0.2 \times \ln 0.2) \\ &+ 0.4 \times (0.3 \times \ln 0.3 + 0.7 \times \ln 0.7)] = 0.5444 \end{aligned} \tag{31}$$

Then, we need to compute the entropy of the predicted belief state:

$$\begin{aligned} H(b(s')) &= -[b(s_1) \times \ln b(s_1) + b(s_2) \times \ln b(s_2)] \\ &= -[0.6 \times \ln 0.6 + 0.4 \times \ln 0.4] = 0.6730 \end{aligned} \tag{32}$$

Finally, the EFE is:

$$F(a) = 0.4297 + 0.5444 - 0.6730 = 0.3011 \tag{33}$$

The agent would compare $F(a)$ across all possible actions and select the one with the lowest free energy.

In real-world systems, scheduling decisions are subject to latent stochastic uncertainties (for example, cache thrashing and memory timing). These phenomena are intermittent, probabilistic, and sometimes not even directly observable, yet they manifest indirectly through measurable performance indicators like execution time and cache miss rate. Their effects are encoded in the empirical covariance matrix $\mathbf{\Sigma}_{s,a}$ of our multivariate Gaussian observation model. While a single Gaussian distribution does not explicitly capture heavy-tailed behaviour, the Active Inference framework, grounded in the Free Energy Principle, mitigates this limitation by adapting inference and action selection in response to observation variance. Specifically, we model the observation likelihood as: $O(\mathbf{o} \mid s', a) = \mathcal{N}(\mathbf{o}; \mu_{s',a}, \mathbf{\Sigma}_{s',a})$, where $\mathbf{\Sigma}_{s',a}$ reflects empirical uncertainty, including latent disturbances such as memory contention or kernel-level interference. As these disturbances increase, $\mathbf{\Sigma}_{s',a}$ broadens, flattening the likelihood function (see Equation (9)) and down-weighting outliers via the Mahalanobis distance. This results in gentler belief updates because posterior beliefs become less peaked, reflecting heightened epistemic uncertainty. In essence, the agent thus becomes more cautious in its inferences, avoiding overreaction to rare spikes in performance. Additionally, the expected surprise component of the free energy, $D_{\mathrm{KL}}\left[b(\mathbf{o} \mid s') \,\|\, P(\mathbf{o} \mid s', a)\right]$ becomes less sensitive to high-variance fluctuations, thereby reducing the influence of infrequent but extreme events. This smoothing effect discourages overly conservative, worst-case reasoning. Instead, minimising EFE yields decisions that balance utility-driven action with uncertainty-aware exploration, making this approach well-suited to dynamic and heterogeneous multi-core environments.

V. Case Study

To assess the effectiveness and robustness of our proposed POMDP Active Inference model for dynamic task-to-core allocation, we conducted case studies using benchmarks on different platform configurations (focused on CPU-based systems) and various uncertainty conditions in order to demonstrate the model's adaptability, performance benefits, and generalisation capabilities in realistic computing environments.

Our evaluation uses a two-platform setup: a reasoning host (MacBook with Apple M1 processor) generates task-to-core mappings, while a Jetson Orin target executes these mappings using fixed scripts. We use the term "script" broadly to denote any predefined execution specification, whether in the form of shell commands, ROS2 launch files, or statically configured RTOS task declarations. This allows us to define the execution order, CPU affinity, and launch configuration for each task instance, thereby enabling a controlled evaluation of decision logic independently of OS integration complexity. During training, task sequences were randomly mapped to cores to expose diverse interference patterns. This diversity enables the model to capture latent platform behaviour. During evaluation, we simulate dynamic conditions by varying task periods and sequences, and apply the trained model to generate optimised task-to-core mappings. This allows us to assess how well the method generalises and outperforms baseline approaches.

A. Experimental Setup

We used the EEMBC CoreMark-Pro benchmark suite [46], [47] as the basis for our experiments. This suite includes a

diverse set of workloads with varying computational characteristics, ensuring comprehensive coverage of task types relevant to modern multi-core systems. All workloads from the CoreMark-Pro suite were included to provide a thorough assessment of the model's capabilities.

1) Platform Configurations: In this case study, we evaluate our model on hardware platforms that, while heterogeneous, still support task migration—necessitating an effective task-to-core mapping strategy. Specifically, we used the NVIDIA Jetson Orin platform and configured its CPU cores to simulate both homogeneous and heterogeneous multi-core environments. This setup allows us to demonstrate the applicability of our approach across a range of system configurations.

Homogeneous Quad-Core Configuration: Four ARM CPU cores with identical processing capabilities on the Jetson Orin platform to create a homogeneous testbed. This setup serves as a baseline scenario, providing insight into our model's behaviour in environments with uniform computational resources and consistent performance characteristics.

Heterogeneous Multi-Core Configuration with Core Speed Variations: The Jetson Orin platform with a mix of CPU cores operating at different frequencies—specifically, 2 high-speed cores running at 2201.6 MHz and 2 normal-speed cores at 1728.0 MHz. This setup emulates a typical non-uniform hardware environment and evaluates the model's ability to adapt task scheduling strategies to varying processing capabilities. It enables us to assess the robustness and flexibility of our framework, demonstrating its suitability for both homogeneous and heterogeneous multi-core systems.

2) Training Data Generation and Model Training: During the model training phase, we collected performance data using 500 randomly generated task release orders, each configured with predefined fixed task periods, which are also randomly selected. It is important to clarify that these period settings were used solely for the purpose of generating diverse training data and were randomly assigned to simulate a wide range of execution patterns. This does not imply any strong assumptions about task execution behaviours or static system configurations. In the subsequent evaluation phase, task periods are intentionally varied to assess the model's adaptability to dynamic system conditions. Notably, if the period of a task during evaluation exceeds the observation window, the task effectively exhibits sporadic-like behaviour from the model's perspective, further challenging its ability to generalise beyond the training scenarios. In our case study, the normalised system utilisation is approximately 1.6, highlighting the significant impact of shared resource contention within the system. These release orders were designed to simulate diverse task arrival patterns and system states, enabling the model to learn from a broad range of scenarios and effectively capture the underlying system dynamics. However, when compared to the kinds of datasets required by deep-learning-based approaches, the dataset is small, not well-labeled, and does not exhaustively cover all possible combinations. This poses challenges for learning methods that rely purely on data, such as deep neural networks (DNNs), and underscores the importance of the model's ability to acquire and generalise "knowledge" beyond the provided dataset.

B. Evaluation Scenarios

To evaluate the model's performance under various conditions, we designed three distinct scenarios that incorporate different uncertainties and system utilisation levels.

Scenario 1 Varying Release Orders with Fixed Periods (Seen Tasks): Assesses the model's robustness to different task arrival patterns using the same set of tasks as in the training data. We randomly generated 100 new release orders, different from those used in the training dataset. The periods of the tasks were kept the same as in the training dataset. This tests the model's ability to generalise its learned allocation strategies to new scheduling conditions without retraining.

Scenario 2 Varying System Utilisation Levels: Evaluates the model's performance under different system load conditions by altering task periods to achieve varying levels of system utilisation. We randomly generated new release orders for the tasks. The periods of the tasks were modified to achieve normalised system utilisation levels of around 0.6, 0.9, and overloaded 1.6. By varying utilisation we evaluate the model's ability to effectively manage resource allocation across a range of load scenarios, ensuring consistent performance despite fluctuating load demands.

Scenario 3 Handling Unseen Tasks: To test the model's generalisation and its ability to handle unseen tasks not included in the training dataset, we created new tasks based on the CoreMark-Pro workloads by randomly increasing the number of times each workload is executed, altering their computational demands. The periods for each of these new tasks were randomly defined to introduce additional variability. This scenario evaluates the effectiveness of our few-shot learning-inspired approach for handling unseen tasks, demonstrating the model's ability to identify similarities with known tasks and make informed allocation decisions without prior specific training data.

C. Comparative Methods and Evaluation Metrics

We evaluated the performance of our model against four other methods: **Worst Fit, Random Fit, Random Forest Regression**, and **k-Means Clustering**. Worst Fit and Random Fit are heuristic methods that provide baseline performance levels. Random Forest and k-means were chosen as representatives of traditional machine learning approaches for regression-based prediction and clustering, respectively.

To quantitatively assess the model's performance, we employed the following metrics:

- **Load Balance:** The workload distribution across cores, evaluating the model's effectiveness in preventing bottlenecks and ensuring smooth system operation. To quantify load balance, we calculated the standard deviation of the core loads relative to the mean core load.
- **Execution Time Proportion to Reference:** The ratio of each task's execution time to its reference execution time (ideal execution time measured by executing the task alone without any interfering tasks), as Equation (34) shows. This metric assesses the impact of interference and

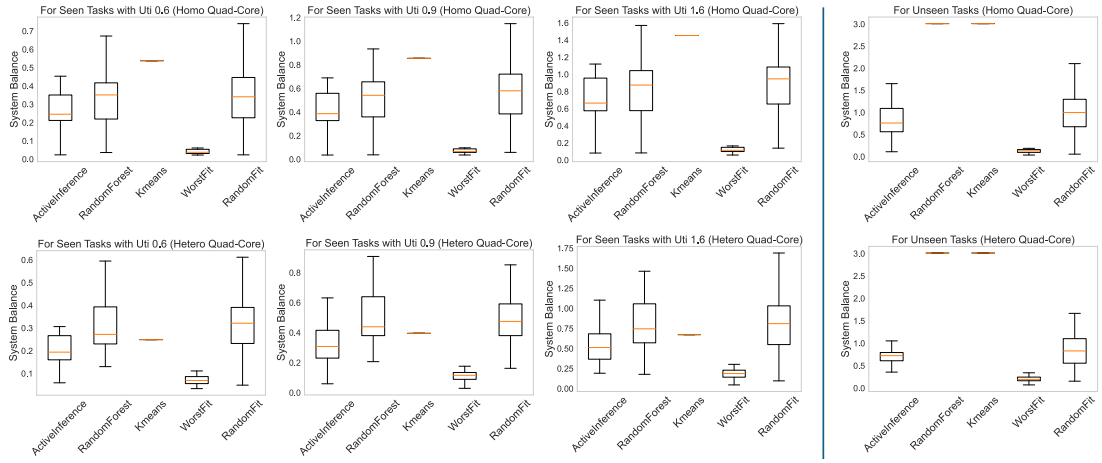

Fig. 3. The balance of the system (i.e. how evenly workload is distributed across the cores. Worst fit is the baseline, and the other approaches can be seen to deviate from a naive worst fit allocation to optimise for other factors (average execution time, criticality, etc.)

the effectiveness of the model in minimising execution time overhead.

$$Ratio = Workload(ET)/Workload(ET_{ref}) \quad (34)$$

- **Deadline Missing Rate:** The percentage of tasks that fail to meet their deadlines, assessing the model's ability to schedule tasks effectively.

TABLE I
INFERENCE TIME PER TASK (IN MILLISECONDS) ACROSS DIFFERENT ALLOCATION STRATEGIES

Method	Mean (ms)	Median (ms)	Std. Deviation (ms)	Max (ms)
Active Inference (POMDP)	0.97	0.94	0.11	2.38
Random Forest	1.82	1.79	0.35	2.67
K-Means Clustering	1.84	1.37	8.58	3.26
Worst-Fit Heuristic	0.00107	0.000954	0.00799	0.30
Random-Fit Heuristic	0.000535	0.000528	0.00230	0.076

D. Inference Time Analysis

Table I presents per-task inference latency across five allocation strategies. The Active Inference (POMDP) method achieves a mean latency of 0.97 ms, with a maximum of 2.38 ms and a low standard deviation of 0.11 ms, indicating stable and bounded responsiveness. Despite involving probabilistic reasoning and belief updates, Active Inference demonstrates efficiency suitable for real-time embedded scheduling. For comparison, the Random Forest model shows higher latency (mean: 1.82 ms; max: 2.67 ms), reflecting the cost of ensemble evaluations. K-Means Clustering exhibits the highest variability (std: 8.58 ms; max: 3.26 ms), making it less suitable for time-critical applications. In contrast, Worst-Fit and Random-Fit Heuristics are extremely fast (mean: 1.07 μs and 0.53 μs, respectively), though their static nature limits adaptability to workload or platform dynamics. While worst-case latencies are reported for completeness, they are dominated by rare long-tail outliers. To better assess practical performance, we estimate the probability that inference time exceeds 1.5 ms, a soft real-time threshold. Assuming inference times are normally distributed with the reported means and standard deviations, we compute exceedance probabilities using the standard normal cumulative distribution function (CDF). Under this model, Active Inference exceeds 1.5 ms in only 0.001% of cases, Random Forest in 81.9%, K-Means Clustering in 51.6%, while both heuristic methods exhibit effectively 0% exceedance. These results highlight that Active Inference strikes a strong balance: sub-millisecond average latency, tight variability, and robust real-time predictability, despite its use of belief-driven, probabilistic reasoning. Finally, we emphasise that the reported inference time reflects only the reasoning loop in Python on an Apple M1 MacBook and excludes system-level overheads such as kernel scheduling or context switches. Actual latency may vary across hardware platforms, and significantly lower inference times are expected from a compiled implementation (e.g., C/C++) or integration into a OS kernel.

E. Load Balance Evaluation Results

As depicted on the left side of Figure 3, for seen tasks, Worst Fit consistently achieves the best system balance on both homogeneous and heterogeneous platforms. This outcome stems from the algorithm's inherent design, which assigns tasks to the least loaded cores, promoting even workload distribution and minimising resource contention. However, this theoretical advantage is undermined by significant practical limitations. Worst Fit fails to account for critical system-level factors, such as the interference characteristics between different types of workloads. These interference effects, often reflected in execution time metrics, can profoundly impact overall system performance. The proposed Active Inference methods adopt a holistic approach that prioritises workload interference management and execution time dynamics. This strategic trade-off results in consistently lower median balance values across evaluations. By intentionally sacrificing some degree of workload balance, Active Inference achieves superior real-world performance outcomes, including reduced execution time variability and enhanced adaptability to dynamic system conditions.

For unseen tasks, as illustrated on the right side of Figure 3, the proposed Active Inference methods continue to outperform

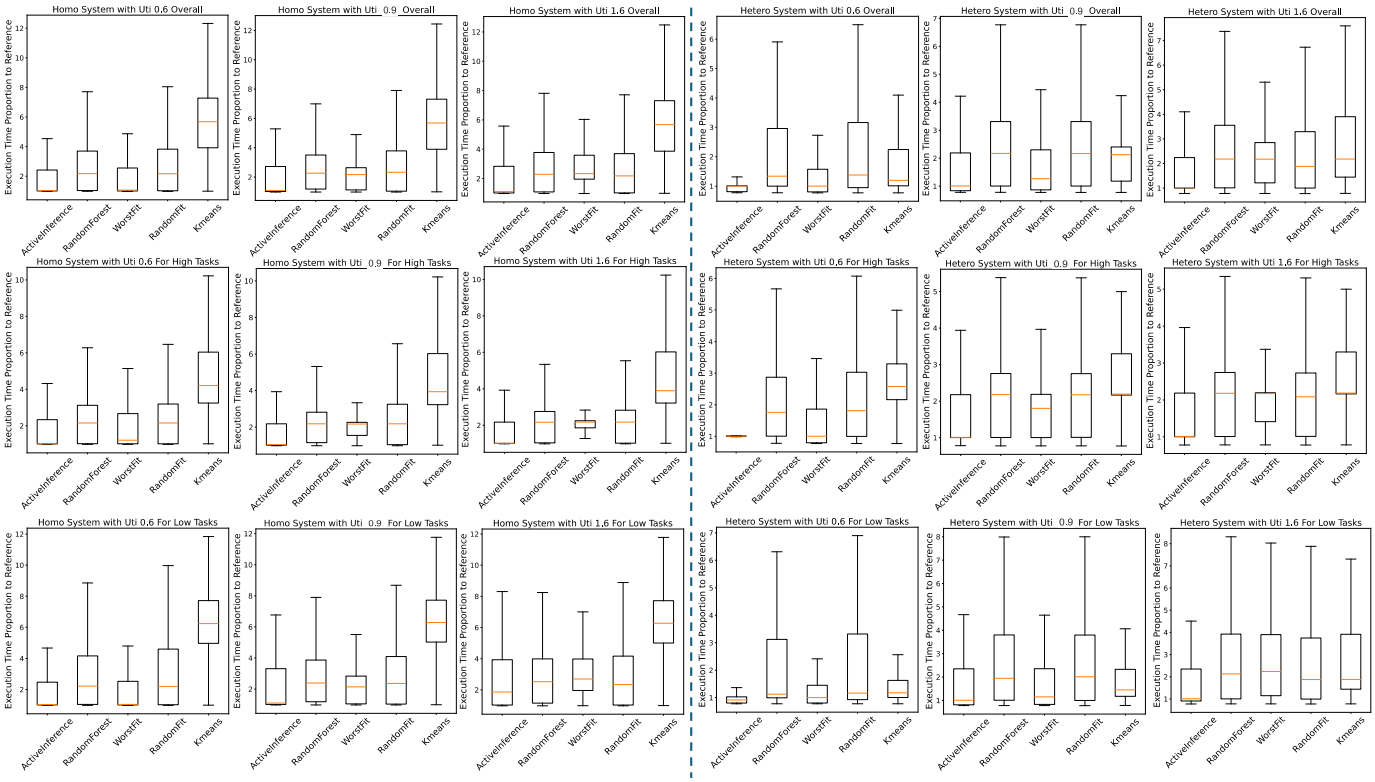

Fig. 4. Execution Time Proportion to Reference of different task allocation algorithms for seen tasks on homogeneous and heterogeneous platforms.

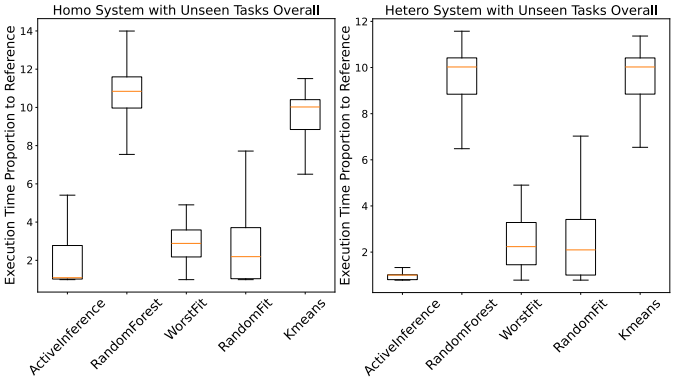

Fig. 5. Execution Time Proportion to Reference of different task allocation algorithms for unseen tasks on homogeneous and heterogeneous platforms.

most other algorithms (with the exception of Worst Fit) in terms of system balance. On both homogeneous and heterogeneous platforms, Random Forest and K-means exhibit significantly higher balance values compared to other algorithms. This performance disparity arises from their heavy reliance on prior knowledge and patterns derived from seen tasks. Their decision-making processes are overly dependent on historical data, which limits their ability to generalise effectively and results in suboptimal performance when faced with dynamic or unpredictable workloads. This highlights a fundamental limitation of traditional machine learning-based approaches: they are not inherently designed to handle variability or adapt to changing contexts without extensive prior training or external adjustments.

F. Execution Time Ratio Evaluation Results

In this subsection, we evaluate the execution time proportion to reference across different task allocation algorithms for seen tasks under various levels of system utilisation and unseen tasks.

For seen tasks on homogeneous platforms, as illustrated on the left side of Figure 4, and on heterogeneous platforms, as shown on the right side of Figure 4, the proposed Active Inference method achieves the lowest median execution time proportion and the tightest performance bounds across all system workloads. Notably, its median execution time consistently outperforms all other methods under all conditions, showcasing the robustness and efficiency of the approach.

When balancing the prioritisation of high-criticality tasks against fairness in the execution of low-criticality tasks, Active Inference delivers consistently superior performance across varying system workloads. High-criticality tasks always with lower median execution time proportions, tighter upper bounds, and greater stability compared to low-criticality tasks. In contrast, other methods exhibit significant fluctuations, with their performance varying unpredictably under different workload conditions. This stability enhances the efficacy of Active Inference in managing diverse task criticalities.

These findings highlight the effectiveness of the proposed method in balancing task execution across different criticality levels. Moreover, even when confronted with execution patterns outside its training dataset, Active Inference continues to deliver exceptional performance, adapting efficiently to system dynamics. This adaptability provides compelling evidence that Active Inference is well-equipped to handle system dynamism

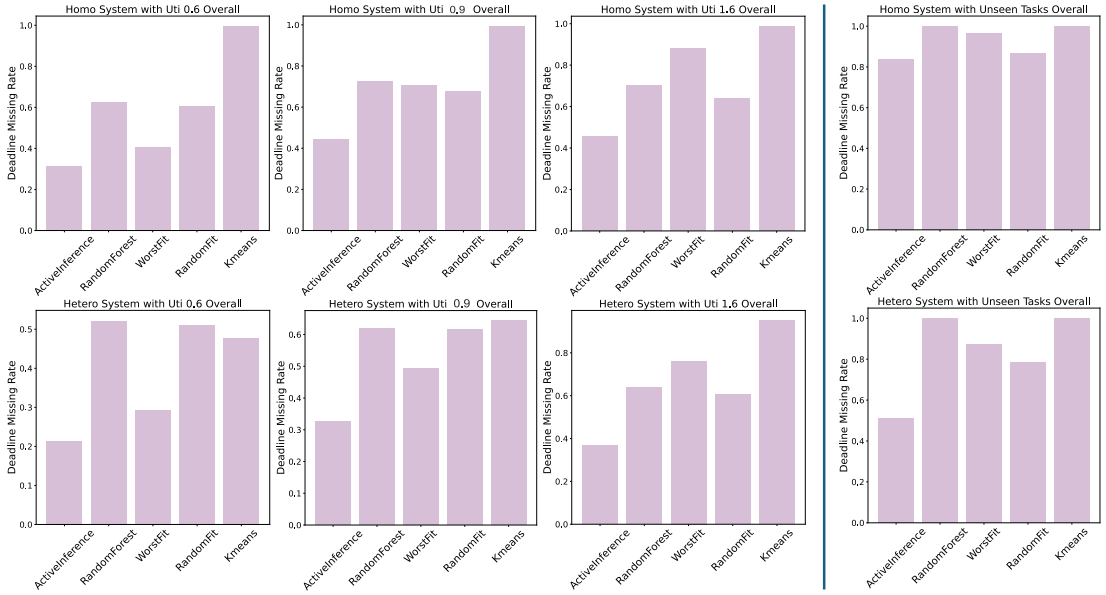

Fig. 6. Deadline Missing Rate across different task allocation algorithms for seen and unseen tasks on homogeneous and heterogeneous platforms.

and maintain efficiency under unpredictable or evolving operational conditions.

For unseen tasks, as illustrated in Figure 5, Active Inference continues to excel. On both homogeneous and heterogeneous platforms, it achieves the lowest median execution time proportion among all methods. These results emphasise Active Inference's adaptability, enabling it to maintain high efficiency even when other approaches falter in generalising to unseen tasks. By consistently delivering low median execution times and demonstrating resilience to variability in workload conditions, Active Inference proves itself as a robust solution for diverse scheduling scenarios in dynamic and unpredictable system environments. It consistently achieves the lowest median execution time proportions across platforms and workload types ensures stable and predictable performance, reinforcing its suitability for real-world dynamic systems.

G. Deadline Missing Rate Results

In this subsection, we evaluate the deadline missing rate. For seen tasks, as illustrated on the left side of Figure 6, Active Inference consistently achieves the lowest deadline missing rates across all scenarios for both homogeneous and heterogeneous platforms. This performance distinguishes it from both heuristic and traditional machine learning-based methods, which struggle to maintain comparable efficiency under similar conditions.

For unseen tasks, as illustrated on the right side of Figure 6, Active Inference demonstrates exceptional adaptability, delivering robust performance even in unpredictable environments. Its ability to consistently maintain the lowest deadline missing rates highlights its resilience and effectiveness in dynamic scenarios. Its consistent ability to achieve the lowest rates across seen and unseen tasks, on both homogeneous and heterogeneous platforms, underscores its capability to ensure optimal system performance under diverse and varying workload conditions.

VI. Conclusion

Strict task isolation is increasingly impractical in modern multi-core systems due to shared resource contention and runtime variability. This work proposes a novel framework that combines POMDPs with Active Inference to enable adaptive, uncertainty-aware task-to-core allocation. By minimising Expected Free Energy (EFE), the method balances performance optimisation with epistemic uncertainty reduction in dynamic, partially observable environments. The framework achieves low-latency inference through workload profiling, dimensionality reduction, and vectorised computation, outperforming conventional learning-based approaches. Evaluated on Jetson Orin CPUs, the Python-based implementation achieves sub-millisecond latency, with further gains expected from compiled or kernel-integrated versions. A hierarchical architecture supports scalability and modular reasoning, laying the groundwork for future extensions including heterogeneous platform support, energy-aware policies, and real-time deadline integration. Overall, the approach offers a practical and principled solution for adaptive scheduling in high-assurance embedded systems.

References

[1] W. M. D. Chia, S. L. Keoh, C. Goh, and C. Johnson, "Risk assessment methodologies for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[2] P. Graydon and I. Bate, "Safety assurance driven problem formulation for mixed-criticality scheduling," *Proc. WMC, RTSS*, pp. 19–24, 2013.

[3] J. Yan and W. Zhang, "Wcet analysis for multi-core processors with shared l2 instruction caches," in *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2008, pp. 80–89.

[4] R. Wagle, Z. Tong, R. L. Sites, and J. H. Anderson, "Want predictable gpu execution? beware smis!"

[5] A. Burns, R. I. Davis, S. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Transactions on Computers*, vol. 67, no. 10, pp. 1478–1491, 2018.

[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[7] D. Kim, H. Park, and J. K. Choi, "Optimal load allocation for coded distributed computation in heterogeneous clusters," *IEEE Transactions on Communications*, vol. 69, no. 1, pp. 44–58, 2020.

[8] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.

[9] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE international real-time systems symposium (RTSS 2007)*. IEEE, 2007, pp. 239–243.

[10] A. Burns and R. I. Davis, "Mixed criticality systems-a review:(february 2022)," 2022.

[11] A. Forootani, R. Iervolino, M. Tipaldi, and J. Neilson, "Approximate dynamic programming for stochastic resource allocation problems," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 4, pp. 975–990, 2020.

[12] X. Xue, X. Ai, J. Fang, W. Yao, and J. Wen, "Real-time schedule of integrated heat and power system: A multi-dimensional stochastic approximate dynamic programming approach," *International Journal of Electrical Power & Energy Systems*, vol. 134, p. 107427, 2022.

[13] A. Burns and R. I. Davis, "Response time analysis for mixed criticality systems with arbitrary deadlines," in *5th International Workshop on Mixed Criticality Systems (WMC 2017)*. York, 2017.

[14] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 78–87.

[15] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 147–152.

[16] R. I. Davis and A. Burns, "Robust priority assignment for fixed priority real-time systems," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. IEEE, 2007, pp. 3–14.

[17] S. K. Baruah, "Mixed-criticality scheduling theory: Scope, promise, and limitations." *IEEE Des. Test*, vol. 35, no. 2, pp. 31–37, 2018.

[18] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2011, pp. 269–279.

[19] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero, "An analyzable memory controller for hard real-time cmps," *IEEE Embedded Systems Letters*, vol. 1, no. 4, pp. 86–90, 2009.

[20] J. Liedtke, H. Hartig, and M. Hohmuth, "Os-controlled cache predictability for real-time systems," in *Proceedings Third IEEE Real-Time Technology and Applications Symposium*. IEEE, 1997, pp. 213–224.

[21] B. C. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson, "Outstanding paper award: Making shared caches more predictable on multicore platforms," in *2013 25th Euromicro Conference on Real-Time Systems*. IEEE, 2013, pp. 157–167.

[22] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Scheduling optimization with partitioning for mixed-criticality systems," *Journal of Systems Architecture*, vol. 98, pp. 191–200, 2019.

[23] A. Agrawal, R. Mancuso, R. Pellizzoni, and G. Fohler, "Analysis of dynamic memory bandwidth regulation in multi-core real-time systems," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 230–241.

[24] Z. Guo, K. Yang, F. Yao, and A. Awad, "Inter-task cache interference aware partitioned real-time scheduling," in *Proceedings of the 35th annual ACM symposium on applied computing*, 2020, pp. 218–226.

[25] D. Griffin, B. Lesage, I. Bate, F. Soboczenski, and R. I. Davis, "Forecast-based interference: Modelling multicore interference from observable factors," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, 2017, pp. 198–207.

[26] S. Zhao, X. Dai, B. Lesage, and I. Bate, "Cache-aware allocation of parallel jobs on multi-cores based on learned recency," in *Proceedings of the 31st International Conference on Real-Time Networks and Systems*, 2023, pp. 177–187.

[27] H. Lee, J. Lee, I. Yeom, and H. Woo, "Panda: Reinforcement learning-based priority assignment for multi-processor real-time scheduling," *IEEE Access*, vol. 8, pp. 185 570–185 583, 2020.

[28] C. Shyalika, T. Silva, and A. Karunananda, "Reinforcement learning in dynamic task scheduling: A review," *SN Computer Science*, vol. 1, no. 6, p. 306, 2020.

[29] E. Jayanthi and R. Vallikannu, "Enhancing the performance of asymmetric architectures and workload characterization using lstm learning algorithm," *Advances in Engineering Software*, vol. 173, p. 103266, 2022.

[30] P. Mohanan and M. Chacko, "A multi objective db-rnn based core prediction and resource allocation scheme for multicore processors," *Computers and Electrical Engineering*, vol. 118, p. 109369, 2024.

[31] D. Iorga, T. Sorensen, J. Wickerson, and A. F. Donaldson, "Slow and steady: Measuring and tuning multicore interference," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 200–212.

[32] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[33] M. Hauskrecht, "Value-function approximations for partially observable markov decision processes," *Journal of artificial intelligence research*, vol. 13, pp. 33–94, 2000.

[34] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces," 2009.

[35] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa, "Online planning algorithms for pomdps," *Journal of Artificial Intelligence Research*, vol. 32, pp. 663–704, 2008.

[36] P. Poupart and C. Boutilier, "Value-directed compression of pomdps," *Advances in neural information processing systems*, vol. 15, 2002.

[37] M. T. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for pomdps," *Journal of artificial intelligence research*, vol. 24, pp. 195–220, 2005.

[38] K. Friston, "The free-energy principle: a unified brain theory?" *Nature reviews neuroscience*, vol. 11, no. 2, pp. 127–138, 2010.

[39] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, G. Pezzulo *et al.*, "Active inference and learning," *Neuroscience & Biobehavioral Reviews*, vol. 68, pp. 862–879, 2016.

[40] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[41] J. Shi and K.-H. Chen, "Shielded reinforcement learning for fault-tolerant scheduling in real-time systems," *Real-Time Systems*, pp. 1–5, 2025.

[42] International Organization for Standardization, "ISO 26262-6:2018 – Road Vehicles – Functional Safety – Part 6: Product Development at the Software Level," https://www.iso.org/standard/68384.html, 2018, accessed June 2025.

[43] AUTOSAR, "AUTOSAR Adaptive Platform – Release R23-11," https://www.autosar.org/standards/adaptive/, 2023, accessed June 2025.

[44] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch, "Mixed-criticality embedded systems–a balance ensuring partitioning and performance," in *2015 Euromicro Conference on Digital System Design*. IEEE, 2015, pp. 453–461.

[45] K. P. Murphy, *Probabilistic machine learning: an introduction*. MIT press, 2022.

[46] J. Poovey *et al.*, "Characterization of the eembc benchmark suite," *North Carolina State University*, vol. 32, pp. 37–50, 2007.

[47] J. A. Poovey, T. M. Conte, M. Levy, and S. Gal-On, "A benchmark characterization of the eembc benchmark suite," *IEEE micro*, vol. 29, no. 5, pp. 18–29, 2009.

Jie Zou is a CfAA Research Fellow in Through-Life Safety of AI. Her recent research has focused on scheduling complex, safety-critical systems, particularly addressing the dynamics of adaptive autonomous systems and the challenges posed by modern multi-core processors.

Ian Gray is an Associate Professor in the Department of Computer Science at University of York. His research interests include real-time systems and their programming models, embedded systems, FPGAs and re-configurable computing, and many/multi-core distributed systems.