SchedulExpert: Graph Attention Meets Mixture-of-Experts for JSSP

Henrik Abgaryan, Tristan Cazenave, and Ararat Harutyunyan

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France

Abstract. The Job Shop Scheduling Problem (JSSP) is a well-known NP-hard problem in combinatorial optimization, where the objective is to optimize job assignments across machines while minimizing a specific criterion such as makespan. Traditional mathematical and heuristic approaches struggle with scalability and handling complex precedence constraints. Recent advancements in artificial intelligence, particularly deep reinforcement learning (DRL) and supervised learning, have shown promise but face challenges such as training instability and reliance on labeled data. To overcome these limitations, we propose **SchedulExpert**, a novel neural architecture based on a Mixture of Experts (MoE) framework with self-supervised learning. SchedulExpert integrates a Graph Attention Network (GAT) encoder, an attention-based routing mechanism, and multiple expert modules to enhance flexibility and efficiency in scheduling decisions. Experimental evaluations on benchmark JSSP instances demonstrate that SchedulExpert achieves competitive performance against state-of-the-art metaheuristic and neural-based methods, offering a scalable and effective solution for real-world scheduling challenges.

Keywords: Job Shop Scheduling, SchedulExpert, GATv2Conv, mixture of experts, self supervised learning

1 Introduction

The Job Shop Scheduling Problem (JSSP) is a complex and extensively researched challenge within production optimization and scheduling. This problem requires determining how to assign N jobs, each characterized by distinct processing durations, to a finite set of M machines. The goal is to improve a chosen performance indicator, such as minimizing the makespan ($C_{\max}$), which represents the total time required to complete all jobs, or optimizing the flow time, defined as the average time each job takes to finish. JSSP has widespread relevance in both manufacturing and service industries, where it influences key outcomes like operational efficiency, effective resource allocation, and the quality of customer service. It is proved that for JSSP instances having more than 2 machines is NP-hard [17]. Therefore, deriving precise solutions for JSSP is generally unfeasible [9]. Historically, methods for addressing the JSSP have been rooted in mathematical programming and heuristic-based strategies [10]. Despite their utility, these approaches often struggle to scale effectively and face challenges in handling intricate job-machine precedence constraints, especially in large-scale scenarios.

Metaheuristic approaches have been widely investigated as a promising alternative to exact methods for addressing the JSSP[25], [11]. Among these, the Priority Dispatching Rule (PDR) is one of the most commonly applied heuristic techniques in practical scheduling systems [39]. While PDRs are popular, designing an effective one is a highly challenging and time-consuming task, requiring significant domain expertise, especially for complex JSSP instances.

Advanced metaheuristics, such as those presented in [27], are capable of producing high-quality solutions within short computational times, often measured in minutes. However, these methods are frequently complex to implement, and their reproducibility can pose significant challenges [4].

The rapid advancements in artificial intelligence (AI) have sparked growing interest in alternative methods to overcome these limitations. Neural networks have shown great potential in finding near-optimal solutions to the JSSP [5], [41]. These learning-based methods can be broadly categorized into two paradigms: supervised learning and reinforcement learning (RL). Deep reinforcement learning (DRL), in particular, has emerged as a promising direction, with research actively developing novel approaches to tackle JSSP [23], [41]. Despite their promise, RL methods are often hindered by challenges associated with unstable training dynamics, making the training process complex and time-intensive [13], [24].

Supervised learning, in contrast to reinforcement learning, avoids many of the instability issues inherent in RL. However, it depends heavily on labeled data, which poses a significant challenge for combinatorial optimization problems like the JSSP. As JSSP is an NP-hard problem [9], generating optimal or near-optimal labels using exact solvers is computationally expensive and often impractical [30]. To address these limitations, semi-supervised learning [28] has gained attention for its ability to utilize unlabeled data effectively. Moreover, self-supervised learning [12], [31]

is emerging as a promising approach for neural combinatorial optimization problems, offering an exciting avenue for advancing research in JSSP and similar domains.

Another promising technique to enhance model performance is the Mixture of Experts (MoE) approach. This method combines multiple expert models, each specializing in distinct aspects of a problem, to achieve superior overall performance [21], [7], [36]. In particular, [7] highlights the effectiveness of MoE in scaling model capacity while incurring minimal computational overhead, making it highly valuable for large-scale machine learning tasks.

Despite these advantages, the application of the Mixture of Experts framework to combinatorial optimization problems, including JSSP, remains limited [18]. Expanding its use in this domain could unlock significant potential for solving complex scheduling and optimization challenges.

Building on these observations, we propose a novel neural mixture of experts architecture, referred to as **SchedulExpert**. **SchedulExpert** is a novel neural mixture of experts architecture designed to tackle JSSP. The proposed architecture is constructed to process graph-structured data efficiently. It begins with the **GATEncoder**, which leverages Graph Attention Networks (GAT) to transform input graph data into meaningful node embeddings, capturing both local and global structural features. To enhance flexibility and model capacity, a **Mixture-of-Experts (MoE)** mechanism is integrated, comprising multiple expert modules, each specializing in processing distinct patterns within the embeddings. A crucial component of the architecture is the **AttentionRouter**, which utilizes an attention mechanism to dynamically assign weights to the outputs of the expert modules. This ensures that only the most relevant experts contribute to the final aggregated embedding, facilitating adaptive and efficient processing. Finally, the **Decoder** (Multi-Head Attention Decoder) decodes the routed embeddings and produces actionable logits. **SchedulExpert** employs a self-supervised training strategy, where the model generates multiple candidate solutions and selects the best one based on the problem objective as a pseudo-label [12].

We validate the effectiveness of **SchedulExpert** on the JSSP, a domain with established benchmarks [34] against numerous baseline methods, both classic algorithms such as metaheuristics and priority dispatching rules but also neural approaches [[41][26][16][29][40][19]]. As recognized in prior works [35], [32], the study of JSSP is pivotal, as it provides foundational insights for addressing more complex variants, such as Dynamic JSSP [35] and Flow Shop Scheduling Problems [32].

The integration of graph attention mechanisms, expert specialization, and self-supervised training makes **SchedulExpert** a highly adaptable and robust framework for JSSP, offering significant potential for advancing scheduling and optimization tasks.

2 Related Work

Deep Learning (DL) has transformed the field of artificial intelligence, leading to innovative solutions across various domains. A notable example is the work by [22], where the authors employed a deep Q-network (DQN) to solve scheduling problems in semiconductor manufacturing, showcasing the potential of Deep Reinforcement Learning (DRL) for industrial applications. Building on this line of research, [41] proposed a DRL-based framework that uses Priority Dispatching Rules (PDRs) for the JSSP. Their method recasts the scheduling task as a Markov Decision Process (MDP) and leverages a disjunctive graph representation to capture both machine states and operation dependencies. To process this representation, they introduced a Graph Isomorphism Network (GIN), whose resulting embeddings feed into a policy network trained with Proximal Policy Optimization (PPO) [33].

Subsequent works have focused on improving the state representation of JSSP to enhance the learning process. For instance, [38] expanded the feature space and adopted a bidirectional scheduling strategy within an MDP framework, reducing the risk of multiple equivalent optimal actions. They also introduced Invalid Action Masking (IAM) to remove infeasible choices, effectively narrowing the search space and guiding the policy toward better solutions. Meanwhile, [14] tackled combinatorial optimization problems (COPs) more generally by modeling them as MDPs and applying Bisimulation Quotienting (BQ) to exploit problem symmetries. This technique not only reduces computational overhead but also improves generalization to larger problem instances. Others [1] have even tried using LLMs end to end for solving JSSP.

Various strategies for sequentially constructing JSSP solutions have been explored. Single-shot (greedy) policies, guided by neural networks, are used in [41, 38], while neural construction methods sample solutions from network probability distributions [3]. Monte-Carlo Tree Search (MCTS) [6] and its variants [8] leverage rollouts but require significant resources. Beam search [15] greedily expands partial solutions, also demanding high computational effort. Randomized Greedy Sampling [2] explores deviations from heuristics to identify better solutions.

A growing branch of literature addresses the cost of expert labels by adopting self-supervised training. For example, [12] introduced a Pointer Network-based generative model that iteratively refines multiple solutions and selects the best one as a pseudo-label. This method outperforms many heuristics and DRL approaches on standard JSSP benchmarks, all without relying on expensive ground-truth solutions. A crucial factor in these self-improvement schemes is the method used to generate pseudo-labels. Specifically, [31] presented a problem-independent sequence decoding approach aimed at boosting solution diversity. By sampling without replacement and excluding previously generated sequences, their method broadens the search space and consistently enhances solution quality. Strong experimental results on the Traveling Salesman Problem (TSP) and Capacitated Vehicle Routing Problem (CVRP) underscore its robustness.

Our work draws inspiration from these self-supervised approaches [12, 31] while integrating additional ideas to handle JSSP more effectively. We propose a Mixture of Experts neural network architecture combined with a sampling-based pseudo-labeling process, similar to [12]. By unifying insights from these studies, our method aims to achieve higher performance, scalability, and flexibility in solving complex scheduling tasks.

3 Preliminary

The Job-Shop Scheduling Problem (JSSP) is formally defined as a problem involving a set of jobs J and a set of machines M. The size of the JSSP problem instance is described as $N\_J \times N\_M$, where N_J represents the number of jobs and N_M the number of machines. For each job $J_i \in J$, it must be processed through n_i machines in a specified order $O_{i1} \rightarrow \ldots \rightarrow O_{in_i}$, where each O_{ij} (for $1 \leq j \leq n_i$) represents an operation of J_i with a processing time $p_{ij} \in \mathbb{N}$. This sequence also includes a precedence constraint. Each machine can process only one job at a time, and switching jobs mid-operation is not allowed. The objective of solving a JSSP is to determine a schedule, that is, a start time S_{ij} for each operation O_{ij}, to minimize the makespan $C_{\max} = \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\}$ while meeting all constraints. The complexity of a JSSP instance is given by $N\_J \times N\_M$.

Furthermore, a JSSP instance can be represented through a disjunctive graph, a concept well-established in the literature [41]. Let $O = \{O_{ij}|\forall i, j\} \cup \{S, T\}$ represent the set of all operations, including two dummy operations S and T that denote the starting and ending points with zero processing time. A disjunctive graph $G = (O, C, D)$ is thus a mixed graph (a graph consisting of both directed edges (arcs) and undirected edges) with O as its vertex set. Specifically, C comprises of directed arcs (conjunctions) that represent the precedence constraints between operations within the same job, and D includes undirected arcs (disjunctions) connecting pairs of operations that require the same machine. Solving a JSSP is equivalent to determining the direction of each disjunctive arc such that the resulting graph becomes a Directed Acyclic Graph (DAG) [41]. Markov Decision Process Formulation (MDP), state representation, action space, state transition follow the methods described in [41]. An action $a_t \in A_t$ is an eligible operation at decision step t.

4 Mixture of Experts with Attention-Based Routing

In this section, we present a details of the Mixture of Experts (MoE) component SchedulExpert neural architecture. Unlike standard deep networks where all parameters are activated for every sample, the MoE framework employs specialized *experts*, each responsible for handling a subset of the input space. An *attention-based router* then dynamically selects (or weights) which expert(s) to engage for a given input, enabling more efficient and specialized computation.

The MoE component consists of:

- **Expert Modules**: Multiple two-layer MLPs, each trained to specialize in a particular subspace of the data. An MLP(multilayer perceptron) is a function approximator that maps input vectors to output vectors through a series of affine transformations followed by nonlinear activation functions, organized in multiple layers.
- **Attention Router**: A learnable mechanism that calculates a per-input weight vector over the experts, indicating how much each expert should contribute.

4.1 Expert Modules

Let $\{\text{Expert}_i\}_{i=1}^{M}$ be the set of M expert modules, each parameterized by a two-layer MLP. For an input feature vector $\mathbf{x} \in \mathbb{R}^{D_{\text{in}}}$, the i-th expert outputs:

$$\mathbf{z}_i^{(1)} = \sigma\big(\mathbf{W}_i^{(1)}\,\mathbf{x} + \mathbf{b}_i^{(1)}\big), \tag{1}$$

$$\mathbf{z}_i^{(2)} = \mathbf{W}_i^{(2)}\,\mathbf{z}_i^{(1)} + \mathbf{b}_i^{(2)}, \tag{2}$$

where $\mathbf{W}_i^{(1)} \in \mathbb{R}^{H \times D_{\text{in}}}$ and $\mathbf{W}_i^{(2)} \in \mathbb{R}^{D_{\text{out}} \times H}$ are weight matrices, $\mathbf{b}_i^{(1)} \in \mathbb{R}^{H}$ and $\mathbf{b}_i^{(2)} \in \mathbb{R}^{D_{\text{out}}}$ are bias terms, and $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU). We denote the final output of expert i by:

$$\mathbf{z}_i = \text{Expert}_i(\mathbf{x}) \;=\; \mathbf{z}_i^{(2)} \;\in\; \mathbb{R}^{D_{\text{out}}}. \tag{3}$$

Thus, each expert produces a D_{out}-dimensional representation of the input $\mathbf{x}$.

4.2 Attention-Based Router

The router is designed to compute a *distribution* over the M experts for each input, indicating how much each expert will contribute to the final output. Let $\mathbf{e} \in \mathbb{R}^{D_{\text{embed}}}$ be an embedding (from a GAT encoder) that we wish to route through the experts. The routing proceeds in three stages: (1) *Multi-Head Self-Attention*, (2) *Expert Score Calculation*, and (3) *Weighted Aggregation*.

Multi-Head Self-Attention The router first produces query, key, and value vectors from $\mathbf{e}$:

$$\mathbf{Q} = \mathbf{W}_Q\,\mathbf{e}, \quad \mathbf{K} = \mathbf{W}_K\,\mathbf{e}, \quad \mathbf{V} = \mathbf{W}_V\,\mathbf{e}, \tag{4}$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{H_{\text{attn}} \times D_{\text{embed}}}$ are learnable matrices. We treat each of these as having a *sequence length* of 1 for simplicity (since we have a single embedding vector per sample), but conceptually this can be extended to a set of embeddings.

The multi-head attention output $\mathbf{a}$ is computed by:

$$\mathbf{a} = \text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}), \tag{5}$$

where $\text{MHA}(\cdot)$ denotes multi-head attention:

$$\mathbf{a} = \big[\text{head}_1 \,||\, \text{head}_2 \,||\, \ldots \,||\, \text{head}_h\big]\,\mathbf{W}_O, \tag{6}$$

$$\text{head}_j = \text{softmax}\Big(\frac{\mathbf{Q}_j \mathbf{K}_j^{\top}}{\sqrt{d_j}}\Big)\,\mathbf{V}_j, \tag{7}$$

with h denoting the number of heads, $d_j = H_{\text{attn}}/h$ the dimension of each head, and $\mathbf{W}_O \in \mathbb{R}^{H_{\text{attn}} \times H_{\text{attn}}}$ an output projection matrix. After attention, we apply normalization (i.e. LayerNorm) to stabilize the training:

$$\mathbf{a}_{\text{norm}} = \text{LayerNorm}\big(\mathbf{a}\big). \tag{8}$$

Expert Score Calculation Next, the router generates expert-specific scores from the attention output:

$$\mathbf{s} = \mathbf{W}_S\,\mathbf{a}_{\text{norm}} + \mathbf{b}_S \;\in\; \mathbb{R}^{M}, \tag{9}$$

where $\mathbf{W}_S \in \mathbb{R}^{M \times H_{\text{attn}}}$ and $\mathbf{b}_S \in \mathbb{R}^{M}$ are learnable parameters. We then apply a softmax function to obtain a probability distribution over the M experts:

$$\mathbf{w} = \text{softmax}(\mathbf{s}) \;\in\; \mathbb{R}^{M}, \tag{10}$$

where w_i is the weight assigned to expert i.

Weighted Aggregation Each expert i is now given the (same) attentively processed input $\mathbf{a}_{\text{norm}} \in \mathbb{R}^{H_{\text{attn}}}$ to produce its own output:

$$\mathbf{z}_i = \text{Expert}_i\big(\mathbf{a}_{\text{norm}}\big), \tag{11}$$

where $\mathbf{z}_i \in \mathbb{R}^{H_{\text{attn}}}$ (assuming $D_{\text{in}} = D_{\text{out}} = H_{\text{attn}}$ for simplicity).

Finally, the MoE output is a weighted sum of these expert outputs:

$$\mathbf{z}_{\text{routed}} \;=\; \sum_{i=1}^{M} w_i\, \mathbf{z}_i \quad \in \quad \mathbb{R}^{H_{\text{attn}}}. \tag{12}$$

This $\mathbf{z}_{\text{routed}}$ is then the *routed embedding*, which then can be used by the Decoder.

4.3 Incorporation into the GAT Pipeline

In our GAT-based encoder-decoder framework for JSSP, the MoE module is placed *after* the final GAT layer. The encoder takes JSSP instance represented as disjunctive graph with 11 hand-crafted features similar to [12] and produces embedding representation. We use the fiollowing hand-crafted features:

1. $C_{o(t,j)-1}(\pi_{<t})$ minus the completion time of machine $\mu_{o(t,j)}$, representing job j's idle time if scheduled at t.
2. $C_{o(t,j)-1}(\pi_{<t})$ divided by the makespan of $\pi_{<t}$, measuring how close job j is to the makespan.
3. $C_{o(t,j)-1}(\pi_{<t})$ minus the average completion time of all jobs, indicating job j's earliness or lateness.

4-6. The difference between $C_{o(t,j)-1}(\pi_{<t})$ and the 1st, 2nd, and 3rd quartiles of job completion times, reflecting j's relative completion.

7. The completion time of $\mu_{o(t,j)}$ divided by the makespan of $\pi_{<t}$, showing how close $\mu_{o(t,j)}$'s completion is to the makespan.

8. The completion time of $\mu_{o(t,j)}$ minus the average completion time of all machines, quantifying its earliness or lateness.

9-11. The difference between $\mu_{o(t,j)}$'s completion and the 1st, 2nd, and 3rd quartiles of machine completion times, showing its relative completion.

Specifically, we take the output $\mathbf{H}^{(2)} \in \mathbb{R}^{N \times E_{\text{embed}}}$ of the second GAT layer, partition each node's embedding across M experts, and feed these partitioned embeddings to the corresponding experts. Let

$$\mathbf{H}_n^{(2)} \;=\; [\,\mathbf{p}_1 \,||\, \mathbf{p}_2 \,||\, \ldots \,||\, \mathbf{p}_M\,] \;\in\; \mathbb{R}^{E_{\text{embed}}},$$

where $\mathbf{p}_i \in \mathbb{R}^{\frac{E_{\text{embed}}}{M}}$ is the portion of the n-th node's embedding directed to expert i. We then compute expert outputs and apply the attention router to obtain the final mixture-of-experts representation $\mathbf{z}_{\text{routed}}$ for each node. Concretely,

$$\mathbf{z}_i = \text{Expert}_i(\mathbf{p}_i), \quad i = 1, 2, \ldots, M, \tag{13}$$

$$\mathbf{z}_{\text{routed}} = \text{Router}\Big([\,\mathbf{z}_1 \,||\, \ldots \,||\, \mathbf{z}_M]\Big). \tag{14}$$

Then each *Expert*$_i$ processes its own chunk $\mathbf{p}_i$ independently. The aggregated output $\mathbf{z}_{\text{routed}}$ is then concatenated with the original node features $\mathbf{X}_n$ to produce the final node embedding used in subsequent component network Decoder:

$$\mathbf{h}_n = \big[\mathbf{X}_n \,||\, \mathbf{z}_{\text{routed}}\big]. \tag{15}$$

4.4 Decoder Architecture

At each time step t the Decoder generates the probability of selecting each job by leveraging both job embeddings $\mathbf{e}_i$ and solution-related features. The Decoder is composed of two main components:

Memory Network. Generates a state $\mathbf{s}_j \in \mathbb{R}^d$ for each job $j \in J$ from the partial solution $\pi_{<t}$.

First, we extract a context vector $\mathbf{c}_j$ from $\pi_{<t}$ (as in [12]), which encodes handcrafted features indicating the status of job j. These vectors are fed into a Multi-Head Attention layer (MHA) followed by a non-linear projection:

$$\mathbf{s}_j \;=\; \text{ReLU}\Big(\big[\mathbf{c}_j\mathbf{W}_1 \;+\; \text{MHA}_{b\in J}\big(\mathbf{c}_b\mathbf{W}_1\big)\big]\mathbf{W}_2\Big), \tag{3}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are projection matrices. The MHA module accounts for the entire set of jobs J when generating $\mathbf{s}_j$, following the idea in [12].

Classifier Network. Outputs the probability p_j of selecting job j by combining the embedding $\mathbf{e}_{o(t,j)}$ of its *ready* operation with the state $\mathbf{s}_j$ from the memory network. We concatenate $\mathbf{e}_{o(t,j)}$ and $\mathbf{s}_j$, then apply a feed-forward network (FNN):

$$z_j \;=\; \mathrm{FNN}\Big(\big[\mathbf{e}_{o(t,j)} \,\|\, \mathbf{s}_j\big]\Big). \tag{4}$$

The scalar $z_j \in \mathbb{R}$ is then transformed via softmax to obtain the probability p_j:

$$p_j \;=\; \frac{e^{z_j}}{\sum_{b\in J} e^{z_b}}.$$

Finally, the decision on which job to select at time t is made by sampling from these probabilities.

4.5 Benefits and Intuition

Parameter Efficiency. Since each input primarily activates only a subset of experts (as determined by $\mathbf{w}$), the network can scale by adding more experts without increasing the per-example computation linearly. Compared to the method described in [12], our proposed ScheduleExpert Neural architecture requires fewer parameters while achieving nearly equivalent performance as seen in table 2. Specifically, with the same hyperparameter configurations (i.e., input_size, hidden_size, embed_size), the ScheduleExpert encoder contains 158,084 parameters, compared to 166,784 in [12]—a reduction of nearly 8,700 parameters or 5.22%. Similarly, under the same settings for context size and hidden dimensions, our decoder has 173,057 parameters compared to 210,177 in [12], resulting in a reduction of 37,120 parameters, or 17.66% less than [12].

Specialization. Each expert can learn to handle different regimes or structures in the JSSP environment (e.g., certain job-machine patterns), improving generalization.

Dynamic Routing. By leveraging the attention router, the model learns to *adaptively* choose which experts to trust for a given embedding, offering flexibility and expressive capacity beyond static parameter sharing.

5 Putting It All Together

When integrated with the encoder and the decoder, the MoE module serves as a *critical* flexible layer that selectively refines the learned node embeddings. Formally, for each node n:

1. **GAT Layers:**
$$\mathbf{H}^{(1)} = \mathrm{GAT}_1(\mathbf{X}, \mathbf{E}), \quad \mathbf{H}^{(2)} = \mathrm{GAT}_2\big([\mathbf{X} \,||\, \mathbf{H}^{(1)}], \mathbf{E}\big).$$
2. **MoE Module:**
$$\mathbf{z}_{\text{routed}}, \mathbf{w} = \mathrm{Router}\Big(\mathbf{H}_n^{(2)}\Big), \quad \mathbf{h}_n = [\,\mathbf{X}_n \,||\, \mathbf{z}_{\text{routed}}\,].$$
3. **MHA Decoder:**
$$\mathrm{logits}_n = \mathrm{MHADecoder}\big(\mathbf{h}_n, \mathbf{s}_n\big).$$

Here, $\mathbf{s}_n$ is any additional context state the decoder might consume (i.e., partial schedules or machine statuses). Then logits_n are used to make scheduling decisions.

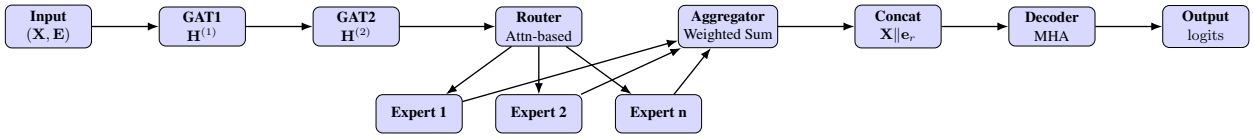

Fig. 1. SchedulExpert architeture pipeline.

5.1 Training Details

The entire MoE module is end-to-end trainable via backpropagation, following the training procedure described in [12]. The training utilizes the cross-entropy loss function. For each training problem instance, S solutions are generated. The makespan is then calculated for each of the generated solutions, and the solution with the minimum makespan is selected as the pseudo-label. As training progresses, the quality of the pseudo-labels improves with the increasing number of training steps. The training was conducted on Nvidia A6000 GPU with 48 GB of RAM. The total training

Fig. 2. Average Gap in percentages during the training and validation

consumed around 35GB of VRAM. To train the model, we used a dataset consisting of 30 000 instances as descussed in [12]. The dataset consists of randomly generatd instances of the following sizes 10 × 10, 15 × 10, 15 × 15, 20 × 10, 20 × 15, 20 × 20 with 5000 instances per each sinstance shape. We train the entire architecture, comprising the GAT encoder, MoE module, and MHA decoder, in an end-to-end manner using mini-batch stochastic gradient descent with the Adam optimizer. Key hyperparameters include an encoder hidden size and output size of 64, determining the dimensions of internal GAT layers and the node embeddings, respectively. Similarly, the memory network hidden size and output size are set to 64, influencing the memory and attention modules in the MHA decoder. The classifier hidden size and latent dimension are also 64, shaping the decoder's classification layers and intermediate embeddings. The MoE router distributes node embeddings across four experts (`n_experts` = 4). Training is performed with a learning rate of 0.0002, a batch size of 128, and over 20 epochs unless early stopping criteria are met. We employ a reducing scheduler (`reduce`) with a minimum learning rate of 10^{-6} to adjust the learning rate based on validation performance. A scaling factor (`beta`) of 512 is used for specialized losses or regularizers. Gradient clipping and other stabilization techniques ensure well-behaved optimization for multi-head attention and MoE routing. All modules are trained simultaneously, with the best model checkpoint selected based on a validation metric after 20 epochs for final evaluation.

6 Experimental Results

To demonstrate the effectiveness of our proposed `SchedulExpert` neural network architecture, we conducted experiments on the well-known TA dataset [34] using a sampling size of 512. Sampling, involves using an already trained neural network to generate multiple solutions and then selecting the best one(having the lowest makespan) from sampled solutions. The performance on each benchmark was evaluated using the percentage *Gap* (G), defined as:

$$\mathrm{G} \;=\; 100 \times \left(\frac{M_{\mathrm{alg}}}{M_{\mathrm{ub}}} \;-\; 1 \right),$$

where M_{alg} represents the makespan generated by the algorithm, and M_{ub} denotes the optimal or best-known makespan for the instance. Lower G values indicate better performance, as they correspond to solutions with objective values closer to the optimal or best-known makespan.

In this section, we present the results and other methods that we used a a baseline comparison.

6.1 Priority Dispatching Rules

We begin by introducing the notations used in these rules, summarized as follows:

$$\begin{aligned}
&Z_{ij}: \text{ the priority index of operation } O_{ij},\\
&n_i: \text{ the number of operations for job } J_i,\\
&Re_i: \text{ the release time of job } J_i \big(\text{here we assume } Re_i = 0 \text{ for all } J_i,\\
&\qquad \text{i.e. all jobs are available in the beginning, but in general}\\
&\qquad \text{the jobs could have different release times}\big),\\
&p_{ij}: \text{ the processing time of operation } O_{ij}.
\end{aligned}$$

Based on the above notations, the decision principles for each baseline are given below:

– **Shortest Processing Time (SPT):**

$$\min Z_{ij} \;=\; p_{ij}.$$

– **Most Work Remaining (MWKR):**

$$\max Z_{ij} \;=\; \sum_{k=1}^{n_i} p_{ik}.$$

– **Most Operations Remaining (MOPNR):**

$$\max Z_{ij} \;=\; n_i - j + 1.$$

– **Taboo search** [26] uses a technique with a specific neighborhood definition that employs concepts of critical paths and blocks of operations.

The evaluation is divided into two categories: Greedy and With Sampling (s = 128). The results indicate that SE-Ours consistently outperforms other methods, achieving the best results across multiple shapes.

In the Greedy setting in table 1, SE-Ours achieves the lowest values in most cases, with particularly strong performance in configurations like 30×15 (18.5) and 30×20 (20.5). The average performance (14.8) confirms its superiority over other methods, which exhibit higher values, such as MWKR (19.5) and MOPNR (20.1). With Sampling setting with $s = 128$ we incorporated $temperature$ and top_k parameters. Through our evaluation we found out that the best values for $top\_k = 0.5$ and $temperature = 0.5$. SE-Ours_512 achieves the best performance across all cases, further improving results compared to SE-Ours_128 and other baselines. The average performance (8.2) demonstrates a substantial improvement over alternative methods, such as RASCAL (10.5) and L2D (14.8). The best results are highlighted in green and with *, confirming that SE-Ours significantly reduces the gap G.

In table 2 we compare our model with different neural methods, including L2S, NeuroLSA, MIP (Mixed Integer Programming), SN, and SE-SE-Ours_512, across various problem shapes. The results indicate that SE-Ours_512 consistently outperforms all other approaches, achieving the lowest values across all cases.

SE-Ours_512 achieves the best performance for every shape, with significant improvements over the competing methods. For instance, in the 30×20 case, SE-Ours_512 achieves 12.8, outperforming L2S (17.5) and SN (23.7). Similarly, for 100×20, SE-Ours_512 records an impressive 2.5, significantly better than L2S (7.9) and MIP (11.0). In Figure 3 we compare the average time(on the x-axis) vs the average gap(on the y-axis) for 80 instances in Tailard dataset [34]. For small values of s (e.g., $s = 1$, $s = 4$, $s = 8$), the average gap is high, exceeding 12-15, but computation time remains low. As s increases, the average gap decreases significantly, reaching values close to 8 when $s = 1024$, albeit with an increased computational time of over 7 minutes. But still for single average instance

from the Tai[34] benchmark, it takes about 5 seconds for $s = 1024$. Figure 4 illustrates the relationship between the total parameter count and the average performance gap, comparing SE-Ours and SLJ with sample size $s = 512$ [12] across various instances of the Taillard benchmark dataset [34]. Notably, despite using 17.66% fewer parameters, ScheduleExpert achieves a comparable average performance gap to SLJ[12] across different instance sizes.

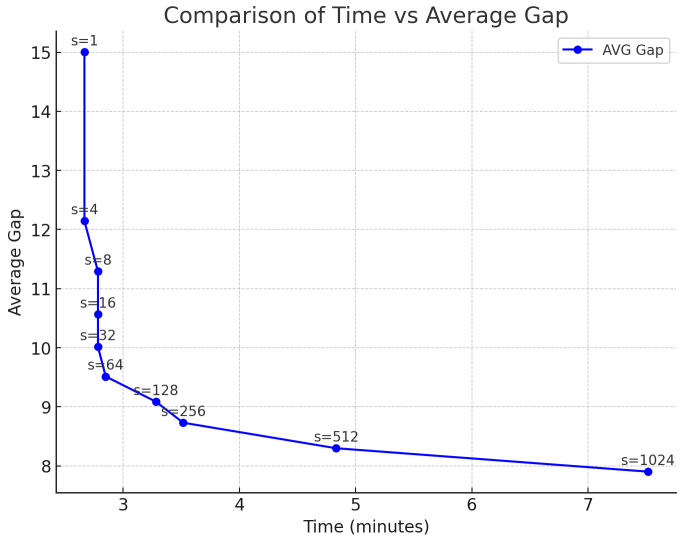

Fig. 3. Required time per different sample size and how it affect on the average gap (the lower the better). Each point corresponds to Average time vs Average Gap in percentages on 80 examples on Tai[34] dataset

6.2 Neural Approaches

– **L2D**[41] models JSSP as an MDP where each step selects an operation, updating a disjunctive graph $G(t)$ and state s_t. The reward $R(a_t, s_t) = H(s_t) - H(s_{t+1})$ guides makespan minimization via policy $\pi(a_t \mid s_t)$. L2D employs a GIN [37] to encode graph-structured data. Node embeddings update iteratively:

$$h_v^{(k)} = \mathrm{MLP}_{\theta_k}\left((1+\epsilon^{(k)})h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right). \tag{16}$$

Global embeddings guide action selection via MLP and softmax. Training uses PPO-based [33] actor-critic with a shared GIN backbone.

– **RASCL** [19] proposes Reinforced Adaptive Staircase Curriculum Learning (RASCL) strategy to enhance reinforcement learning in job-shop scheduling by dynamically adjusting task difficulty and revisiting challenging instances, leading to improved dispatching policies and reduced optimality gaps.

– In **L2S**[40] at first JSSP solution is obtained by GNN, using RL agent it iteratively refines the schedules through a learned improvement heuristic.

– **NeuroLSA**[16] presents GNN-based controller that dynamically selects acceptance criteria, neighborhood operators, and perturbation strategies within a local search framework to effectively solve combinatorial optimization problems.

– **ScheduleNet**[29] uses RL-based decentralized scheduler that coordinates multiple agents to complete tasks with minimal makespan, utilizing a type-aware graph attention mechanism to effectively represent and process scheduling problems.

– We also utilized Mixed Integer Programing as a comparison **MIP**[20] with execution time limit of 1 hour.

Shape	Greedy							With sampling $s = 128$						
	SPT	MWKR	MOPNR	Taboo	L2D	RASCL	SE-Ours	SPT	MWKR	MOPNR	L2D	RASCL	SE-Ours$_{128}$	SE-Ours$_{512}$
15×15	26.1	19.1	20.4	14.4	26.0	**14.3***	14.7	13.5	13.5	12.5	17.1	19.1	7.5	**6.5***
20×15	32.3	23.3	24.9	18.9	30.0	16.5	**14.8***	18.4	17.2	16.4	23.7	16.0	10.0	**9.1***
20×20	28.3	21.8	22.9	17.3	31.6	**17.3***	18.3	16.7	15.6	14.7	22.6	19.0	10.1	**9.3***
30×15	35.0	24.1	22.9	21.1	33.6	18.5	**18.5***	23.2	19.0	17.3	24.4	14.0	11.8	**11.1***
30×20	33.4	24.8	26.2	20.7	36.3	21.5	**20.5***	23.6	19.9	20.4	28.4	16.1	13.9	**12.8***
50×15	24.0	16.4	17.6	15.9	22.4	12.2	**12.2***	14.1	13.5	13.5	17.1	19.3	7.8	**7.0***
50×20	25.6	17.8	16.8	20.3	26.5	**13.2***	14.0	17.6	14.6	14.0	20.4	9.9	8.4	**7.7***
100×20	14.0	8.3	8.7	13.5	13.6	5.9	**5.8***	10.4	7.0	7.1	13.3	4.0	2.6	**2.5***
Avg	**27.4**	**19.5**	**20.1**	**18.0**	**27.1**	**14.9**	**14.8***	**17.2**	**15.0**	**14.5**	**14.8**	**10.5**	**9.0**	**8.2***

Table 1. The average percentage gap (G) comparison of different heuristics and L2D neural methods against our model. In each row, the best (minimum) value is highligted with color green and *. Sampling, involves generating multiple solutions with a fixed method or algorithm and then selecting the best one(having the lowest makespan) from sampled solutions.

Shape	L2S	NeuroLSA	MIP	SN	SE-Ours$_{512}$
15×15	9.3	7.7	0.1	15.3	**6.5***
20×15	11.6	12.2	3.2	19.4	**9.1***
20×20	12.4	11.5	2.9	17.2	**9.3***
30×15	14.7	14.1	10.7	19.1	**11.1***
30×20	17.5	16.4	12.6	23.7	**12.8***
50×15	11.0	11.0	12.2	13.9	**7.0***
50×20	11.8	11.2	13.5	13.5	**7.7***
100×20	7.9	5.9	11.0	6.7	**2.5***
Avg	**12.2**	**11.3**	**8.4**	**16.1**	**8.2***

Table 2. The average percentage gap (G) comparison of different neural methods and mixed integer programming(MIP) against our model. In each row, the best (minimum) value is highligted with color green and *.

7 Conclusion

In this work, we presented SchedulExpert, a neural Mixture of Experts model for efficient JSSP solving. By combining graph-based representations, attention-driven routing, and self-supervised learning, it achieves strong performance with fewer parameters and reduced reliance on labeled data. Experiments show competitive results against traditional and deep learning methods. Future work will extend SchedulExpert to dynamic JSSP and other combinatorial optimization tasks demonstrating its versatility and efficiency.

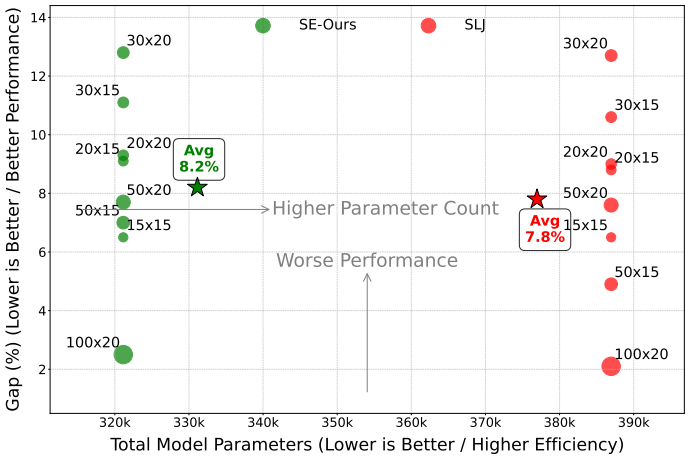

Fig. 4. Average Gap vs. Total Parameter Count on different size instances $(N_J \times N_M)$: SE-Ours with sample size $s = 512$ vs. SLJ[12] on 80 examples on Tai[34] dataset. The average gap across all shape of instances is marked with a star.

References

1. Henrik Abgaryan, Ararat Harutyunyan, and Tristan Cazenave. Llms can schedule, 2024.
2. Henrik Abgaryan, Ararat Harutyunyan, and Tristan Cazenave. Randomized greedy sampling for jssp. In *Proceedings of the 18th Learning and Intelligent Optimization Conference (LION 18)*, volume 14990 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2025.
3. Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2017.
4. Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
5. Giovanni Bonetta, Davide Zago, Rossella Cancelliere, and Andrea Grosso. Job shop scheduling via deep reinforcement learning: a sequence to sequence approach. In *Learning and Intelligent Optimization*, pages 475–490. Springer International Publishing, 2023.
6. Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.
7. Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*, 2024.
8. Tristan Cazenave. Nested Monte-Carlo search. In *Proceedings of the IJCAI International Joint Conference on Artificial Intelligence*, pages 456–461, 2009.
9. Ceren Cebi, Enes Atac, and Ozgur Koray Sahingoz. Job shop scheduling problem and solution algorithms: A review. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7, 2020.
10. S. A. Chaudhry and S. Khan. Comparison of dispatching rules in job-shop scheduling problem using simulation: A case study. *ResearchGate*, 2015.
11. Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms, part ii: Hybrid genetic search strategies. *Computers & Industrial Engineering*, 36(2):343–364, 1999.
12. Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024.
13. Vibhavari Dasagi, Jake Bruce, Thierry Peynot, and Jürgen Leitner. Ctrl-z: Recovering from instability in reinforcement learning. *CoRR*, abs/1910.03732, 2019.
14. Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation Quotienting for Efficient Neural Combinatorial Optimization. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2023)*, 2023.
15. Rupert Ettrich, Marc Huber, and Günther Raidl. *A Policy-Based Learning Beam Search for Combinatorial Optimization*, pages 130–145. Springer, 03 2023.
16. Jonas K. Falkner, Daniela Thyssens, Ahmad Bdeir, and Lars Schmidt-Thieme. Learning to control local search for combinatorial optimization. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, 2022. Accepted for publication.
17. Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.
18. Shibal Ibrahim, Wenyu Chen, Hussein Hazimeh, Natalia Ponomareva, Zhe Zhao, and Rahul Mazumder. Comet: Learning cardinality constrained mixture of experts with trees and local search. *arXiv preprint arXiv:2306.02824*, 2023.
19. Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal Ochoa de Retana, and Martin Takac. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.
20. Wen-Yang Ku and J. Christopher Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:42–53, 2016.
21. Yuanzhi Li and Quanquan Gu. Towards understanding the mixture-of-experts layer in deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
22. Chun-Cheng Lin, Der-Jiunn Deng, Yen-Ling Chih, and Hsin-Ting Chiu. Smart manufacturing scheduling with edge computing using multiclass deep q network. *IEEE Transactions on Industrial Informatics*, 15(7):4276–4284, 2019.
23. Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.
24. Evgenii Nikishin, Pavel Izmailov, Ben Athiwaratkun, Dmitrii Podoprikhin, Timur Garipov, Pavel Shvechikov, Dmitry Vetrov, and Andrew Gordon Wilson. Improving stability in deep reinforcement learning with weight averaging. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI) Workshop on Uncertainty in Deep Learning*, 2018.
25. Eugeniusz Nowicki and Czesław Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.
26. Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

27. Eugeniusz Nowicki and Czesław Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.
28. Yassine Ouali, Céline Hudelot, and Myriam Tami. An overview of deep semi-supervised learning, 2020.
29. J. Park, S. Bakhtiyarov, and J. Park. Schedulenet: Learn to solve multiagent scheduling problems with reinforcement learning, 2022. Preprint.
30. Laurent Perron and Vincent Furnon. Or-tools.
31. Jonathan Pirnay and Dominik G. Grimm. Take a step and reconsider: Sequence decoding for self-improved neural combinatorial optimization. In *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 1927–1934. IOS Press, 2024.
32. Daniel Alejandro Rossit, Fernando Tohmé, and Mariano Frutos. The non-permutation flow-shop scheduling problem: A literature review. *Omega*, 77:143–153, 2018.
33. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
34. Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
35. Libing Wang, Xin Hu, Yin Wang, Sujie Xu, Shijun Ma, Kexin Yang, Zhijun Liu, and Weidong Wang. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Computer Networks*, 190:107969, 2021.
36. Timon Willi, Johan Obando-Ceron, Jakob Foerster, Karolina Dziugaite, and Pablo Samuel Castro. Mixture of experts in a mixture of rl settings. *arXiv preprint arXiv:2406.18420*, 2024.
37. Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.
38. Erdong Yuan, Shuli Cheng, Liejun Wang, Shiji Song, and Fang Wu. Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, 143:110436, 2023.
39. Mohamed Habib Zahmani, Baghdad Atmani, Abdelghani Bekrar, and Nassima Aissani. Multiple priority dispatching rules for the job shop scheduling problem. In *3rd International Conference on Control, Engineering Information Technology (CEIT'2015)*, Tlemcen, Algeria, 2015.
40. Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep reinforcement learning guided improvement heuristic for job shop scheduling. In *International Conference on Learning Representations (ICLR)*, 2024.
41. Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.