Towards a Unified Model for Flexible Job Shop Scheduling with Diverse Constraints

Inguk Choi[1], **Woo-Jin Shin**[2], **Sang-Hyun Cho**[1], **Hyun-Jung Kim**[1*]
[1]Department of Industrial and Systems Engineering, KAIST
[2]School of Industrial and Systems Engineering, Gyeongsang National University
{inguk0826, ie02002, hyunjungkim}@kaist.ac.kr, wjshin@gnu.ac.kr

Abstract

Deep learning-based heuristics have shown great promise for solving the Flexible Job Shop Scheduling Problem (FJSSP). However, existing studies typically overlook operational constraints prevalent in real-world industries or target only a specific FJSSP variant, limiting their practical applicability. In this paper, we introduce SCHEDFORMER, a multi-task scheduling agent that solves FJSSP variants with diverse constraints and generalizes zero-shot to those unseen during training. To support a range of variants within a common framework, we propose a Unified State Representation (USR) that maps different types of constraints into a shared feature space encoding their combined temporal effects, and captures invariant operation-machine relationships across variants via four semantically distinct graphs. To fully exploit the structure of the USR, we also develop a unified neural solver with graph-specific Transformer blocks. Each block employs an attention mechanism specialized for its target graph, while the mapped constraint information conditions the overall representation learning. Experiments on 16 FJSSP variants show that SCHEDFORMER compares favorably with baselines, including specialized neural solvers, and generalizes very well to unseen variants. Further experiments demonstrate its applicability to other scheduling problems.

1 Introduction

Many real-world decision-making problems that require orchestrating limited resources over time can be naturally framed as scheduling problems. Effective scheduling is critical for improving productivity and reducing operational costs, particularly in modern industries such as manufacturing [1], healthcare [2], logistics [3], and cloud computing [4]. In day-to-day operations, scheduling decisions often need to be made quickly in response to frequently changing conditions. Practitioners have therefore long relied on fast heuristics, notably Priority Dispatching Rules (PDRs). Yet designing strong and robust dispatching logic by hand requires substantial domain expertise and trial and error.

Motivation. This bottleneck has spurred a shift toward learning-based PDR design, where deep neural networks automatically learn dispatching policies from data, reducing reliance on handcrafted rules and potentially discovering strategies beyond human intuition [5, 6, 7]. The Flexible Job Shop Scheduling Problem (FJSSP) serves as a general and challenging testbed for such research, encompassing many classical machine environments as special cases [8]. However, most models in the existing literature still lack the versatility that practical scheduling environments demand. Real-world systems involve numerous operational constraints, such as machine maintenance [9], setup times [10], and transportation times [11]. These constraints often coexist in different combinations, yielding diverse problem variants. Many studies nonetheless overlook such constraints, focusing only on the classic FJSSP. More critically, even methods that address specific variants remain single-task solvers,

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

each specialized for one variant [12, 13, 14]. In other words, adapting to a new problem variant often requires redesigning the model and retraining it from scratch, incurring substantial engineering and computational costs. This severely limits their applicability, as scheduling requirements vary widely across industries and from site to site, and may also be revised over time [15]. It is therefore natural to ask: *Can we develop a single neural solver that handles multiple FJSSP variants with different constraint combinations and generalizes to unseen ones without model modification or retraining?*

Contributions. In this paper, we propose SCHEDFORMER, a multi-task scheduling agent that solves diverse FJSSP variants and generalizes in a zero-shot manner to variants not seen during training. SCHEDFORMER comprises three core components. First, we observe that many FJSSP variants can generally be represented in a common language despite differences in their constraints. Building on this insight, we introduce a Unified State Representation (USR) that maps different types of constraints into a shared feature space encoding their combined temporal effects on scheduling, and models invariant operation-machine relationships across variants via four semantically distinct graphs. Second, to effectively leverage the structure of the USR, we develop a unified neural solver composed of multiple Transformer blocks, each equipped with an attention mechanism tailored to the corresponding graph type. We also inject mapped constraint information directly into the core of attention computations, allowing the model to contextualize the constraint effects within the broader scheduling structure. Finally, we train SCHEDFORMER using Adaptive Self-Imitation Learning (ASIL), enabling policy optimization without variant-specific expert labels or reward shaping.

We evaluate SCHEDFORMER on 16 FJSSP variants across multiple instance sizes and data distributions, comparing it with an exact solver, classical PDRs, and neural solvers equipped with our constraint representation. Experiments show that SCHEDFORMER considerably outperforms the PDRs and neural baselines while remaining competitive with the exact solver at a fraction of its runtime. It also achieves state-of-the-art (SOTA) performance among neural solvers on well-known FJSSP benchmarks. Notably, even when applied zero-shot to unseen constraints and their combinations, SCHEDFORMER beats neural baselines trained on all variants, with statistical significance. Finally, we demonstrate effective transfer of SCHEDFORMER to other scheduling problem families.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 formalizes the classic FJSSP and its variants. Section 4 introduces SCHEDFORMER. Section 5 reports experimental results, and Section 6 concludes with limitations and future directions.

2 Related Work

2.1 Deep Learning-Based Heuristics for the FJSSP

Motivated by the remarkable success of Neural Combinatorial Optimization (NCO) [16, 17], deep learning-based heuristics have emerged as powerful tools for tackling the FJSSP [18, 19, 20, 21, 22, 23, 24]. These methods typically represent the construction state as disjunctive or heterogeneous graphs and extract meaningful representations using Graph Neural Networks (GNNs) [25] trained via Reinforcement Learning (RL) algorithms such as REINFORCE [26, 27] or Proximal Policy Optimization (PPO) [28]. For example, HGNN [29] employs a heterogeneous GNN to capture the complex relationships among operations and machines, and directly selects an operation-machine pair at each decision step. DAN [30] refines operation and machine embeddings through a dual graph attention architecture, improving performance on several FJSSP benchmarks. More recently, ReSched [31] proposes a Transformer-based [32] solver with a compact state representation, achieving new SOTA results. Despite these architectural advances, the scope of these methods is primarily limited to the classic FJSSP. Some studies attempt to address variants with specific operational constraints [33, 34, 35], but develop only single-task solvers tailored to one variant at a time. Cross-variant generalization for scheduling problems thus remains largely underexplored.

2.2 Multi-Task Approaches for Combinatorial Optimization (CO) Problems

Recent studies in NCO have shown increasing interest in developing multi-task solvers for CO problems. POMO-MTL [36] models different Vehicle Routing Problem (VRP) variants as combinations of predefined constraints and trains a shared backbone based on the Attention Model (AM) [37]. Building on this combination-based approach, subsequent studies have introduced mixture-of-experts or -depths architectures for improving model capacity while maintaining efficiency [38, 39], a modern

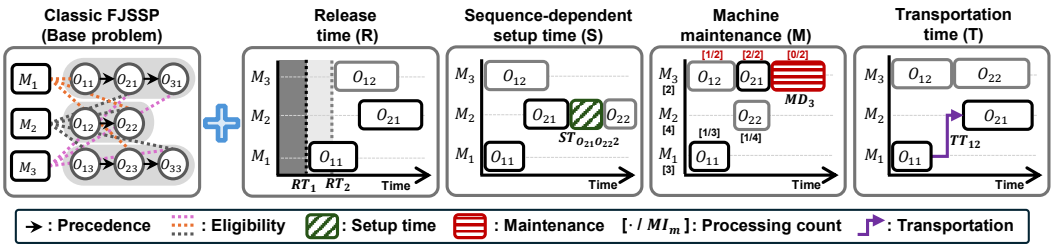


Figure 1: **Illustration of the classic FJSSP and the four operational constraints.** The **left panel** shows the heterogeneous graph representation of the classic FJSSP, while the **four panels on the right** present an example Gantt chart for each constraint.

Transformer-based encoder for enhancing performance and training stability [40], and constraint-aware attention mechanisms for enriching attribute representations [41]. Another paradigm attaches task-specific modules to a shared backbone. AM-FT [42] fine-tunes lightweight adapters on the AM pre-trained for the traveling salesman problem to solve related routing problems. GOAL [43] proposes a multi-type Transformer with one attention block per node-type combination, whose parameters are shared across all blocks. Pre-trained via Supervised Learning (SL), it is adapted to unseen CO problems through problem-specific input and output layers. Our work falls into the combination-based paradigm while substantially broadening its scope from VRPs to scheduling problems. Scheduling problems exhibit graph structures that are fundamentally different from those of VRPs, involving multiple edge types; this calls for new representations and specialized architectures. Moreover, unlike existing combination-based approaches, which require a separate input feature for each constraint, our method unifies diverse constraints into a single form, enabling strong zero-shot generalization to unseen constraints and their combinations without adding new features or fine-tuning.

3 Preliminaries

3.1 Flexible Job Shop Scheduling Problem Variants

We formalize a **classic FJSSP** instance of size $|\mathcal{J}| \times |\mathcal{M}|$ with the following sets: a set $\mathcal{J}$ of jobs, a set $\mathcal{M}$ of machines, and a set $\mathcal{O}$ of operations. Each job $J_j \in \mathcal{J}$ consists of n_j operations (thus $\sum_{j=1}^{|\mathcal{J}|} n_j = |\mathcal{O}|$) that must be processed in a specific order $O_{1j} \to \cdots \to O_{ij} \to \cdots \to O_{n_j j}$ *(Precedence)*. Each operation $O_{ij} \in \mathcal{O}$ can be processed on any machine from its set of eligible machines $\mathcal{M}_{ij} \subseteq \mathcal{M}$ with processing time PT_{ijm} on machine $M_m \in \mathcal{M}_{ij}$ *(Machine eligibility)*. The objective is to minimize the makespan, $C_{\max} = \max_{i,j} C_{ij}$, where $C_{ij} = S_{ij} + PT_{ijm_{ij}}$ denotes the completion time of O_{ij}, S_{ij} its start time, and m_{ij} the index of its assigned machine. In the classic FJSSP, S_{ij} is computed as $\max(C_{(i-1)j},\ AT^t_{m_{ij}})$, where $AT^t_{m_{ij}}$ is the available time of $M_{m_{ij}}$, defined as the completion time of its last operation assigned before step t, or 0 if none exists.

On top of the classic FJSSP, we consider four additional constraints, illustrated in Figure 1, that reflect realistic operational requirements and have been extensively studied in the scheduling literature [44]. **1) Release time (R):** The first operation of each job, O_{1j}, can start only at or after its release time RT_j. **2) Sequence-dependent setup time (S):** If an operation $O_{i'j'}$ is scheduled immediately after O_{ij} on M_m, an additional setup time $ST_{O_{ij}O_{i'j'}m}$ is required before processing can begin. **3) Machine maintenance (M):** Machine M_m requires preventive maintenance for a fixed duration MD_m after completing every MI_m operations. During maintenance, the machine is unavailable for processing any job. **4) Transportation time (T):** When an operation finishes on M_m and its successor is scheduled on a different machine $M_{m'}$ ($m' \neq m$), a transportation time $TT_{mm'}$ is required to transfer the job between the two machines. By combining these constraints, we define 16 FJSSP variants (e.g., FJSSP with S and T $\to$ FJSSP-ST). Further details are provided in Appendix A.

3.2 Autoregressive Solution Construction for FJSSP Variants

For an instance $\boldsymbol{x}$ of any FJSSP variant defined above, a solution $\boldsymbol{\tau}$ can be generated by sequential construction. At each decision step t, a deep autoregressive policy $\pi_\theta(a_t|s_t, \boldsymbol{x})$ outputs a distribution

over the feasible action space $\mathcal{A}_t$, where a_t and s_t denote the action and state at step t, respectively. Here, each action $a_t \in \mathcal{A}_t$ corresponds to an Operation-Machine (O-M) pair e_{ijm}, where O_{ij} is unscheduled, all its predecessors are scheduled, and $M_m \in \mathcal{M}_{ij}$. Selecting $a_t = e_{ijm}$ appends O_{ij} to the end of the current processing sequence of M_m and assigns it the earliest feasible start time S_{ij} under all active constraints. This decision process is repeated until all operations are dispatched, yielding a complete schedule. Thus, a solution $\boldsymbol{\tau}$ can be represented as a sequence of decisions, $\boldsymbol{\tau} = (a_1, \ldots, a_{|\mathcal{O}|})$, and the probability of generating $\boldsymbol{\tau}$ is $\pi_\theta(\boldsymbol{\tau}|\boldsymbol{x}) = \prod_{t=1}^{|\mathcal{O}|} \pi_\theta(a_t|s_t, \boldsymbol{x})$.

4 SCHEDFORMER

We now explain SCHEDFORMER, a unified framework for solving multiple FJSSP variants using a single trained model. We first present the USR, which encodes the states of diverse problem variants in a common format (Section 4.1). We then describe how the unified neural solver processes the numerical and structural information from the USR to guide dispatching decisions (Section 4.2). Finally, we detail the model training procedure (Section 4.3).

4.1 Unified State Representation

The USR is grounded in two properties shared across FJSSP variants: **common temporal effects of constraints** and **invariant relational structures**.

Common Temporal Effects of Constraints. While constraints arise from different operational requirements, they typically have the same effect on scheduling: *they delay when each operation can start and finish on a given machine*. For example, S and M shift machine availability, while R and T postpone operation readiness. Motivated by this observation, we propose a constraint unification method that maps the temporal effects of different constraints into two scalars per O-M pair: **Earliest Start Time (EST)** and **Earliest Finish Time (EFT)**. Concretely, for each candidate action $e_{ijm} \in \mathcal{A}_t$, let δ^t_{ij}, δ^t_m, and δ^t_{ijm} denote the *delays* that the active constraints impose on operation readiness, machine availability, and pair feasibility at step t, respectively. Combining these terms, we define EST^t_{ijm}, the start time of O_{ij} if assigned to M_m at step t, generalizing S_{ij} in the classic FJSSP:

$$\text{EST}^t_{ijm} = \max(\underbrace{C_{(i-1)j} + \delta^t_{ij}}_{\text{Operation readiness}}, \underbrace{AT^t_m + \delta^t_m}_{\text{Machine availability}}) + \underbrace{\delta^t_{ijm}}_{\text{Pair feasibility}} . \quad (1)$$

EFT^t_{ijm} is then given by $\text{EST}^t_{ijm} + PT_{ijm}$. Through this lens, diverse constraints can be interpreted in a common space, with variant changes manifested as shifts in EST/EFT values, like in Figure 2. The EST computation is straightforward and is presented in Appendix B.1. Based on this, we represent each O-M pair by the feature vector $\boldsymbol{f}^t_{e_{ijm}} = [\text{EST}^t_{ijm}, PT_{ijm}, \text{EFT}^t_{ijm}] \in \mathbb{R}^3$. For $e_{ijm} \notin \mathcal{A}_t$, EST^t_{ijm} and EFT^t_{ijm} are set to -1 to mark them as not yet determined. Operation features $\boldsymbol{f}^t_{O_{ij}} \in \mathbb{R}^2$ and machine features $\boldsymbol{f}^t_{M_m} \in \mathbb{R}^4$, which are shared across variants, are described in Appendix B.2. These three types of features provide a numerical representation of the state.

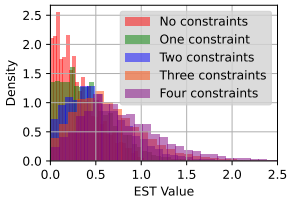


Figure 2: **Histograms of EST values** across 16 variants during testing.

Invariant Relational Structures. Although FJSSP variants differ in their active constraints, their underlying operation-machine structure remains invariant: *bidirectional eligibility* between operations

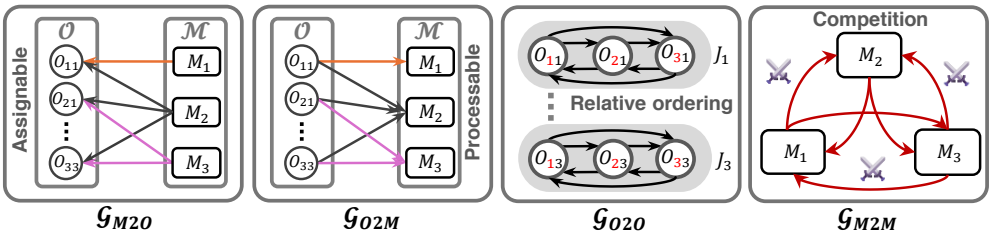


Figure 3: **Illustration of the four graphs.** Each arc indicates the direction of message passing.

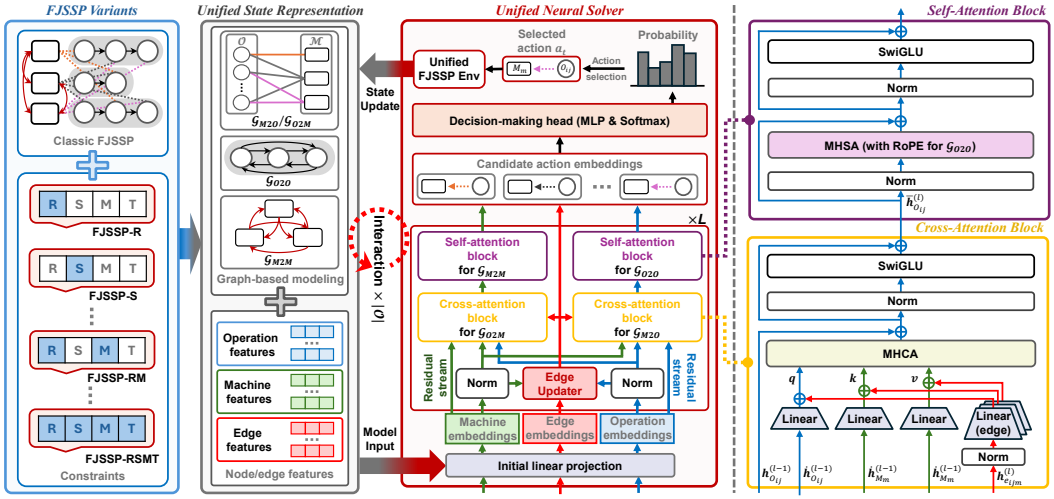


Figure 4: **Overview of the SCHEDFORMER framework.** The USR casts every FJSSP variant into the same input space, from which the unified neural solver makes dispatching decisions.

and machines, *relative ordering* among operations, and *competition* among machines over remaining operations. To explicitly model these semantic relationships, we define four distinct graphs, as illustrated in Figure 3, where operations and machines are represented as nodes with features. **1)** $\mathcal{G}_{M2O}$ identifies which machines are eligible for each operation. **2)** $\mathcal{G}_{O2M}$ specifies which operations each machine can process. **3)** $\mathcal{G}_{O2O}$ encodes the intra-job ordering structure of operations. **4)** $\mathcal{G}_{M2M}$ represents contention among machines for unscheduled operations. The O-M pair features $\boldsymbol{f}_{e_{ijm}}^{t}$ are used as edge features in $\mathcal{G}_{M2O}$ and $\mathcal{G}_{O2M}$, while $\mathcal{G}_{O2O}$ and $\mathcal{G}_{M2M}$ model only the relations between nodes (with self-loops). Together, these graphs provide a structural view of the state.

By expressing all variants using a single language, we can flexibly add, remove, or recombine constraints while enabling the policy to learn universal dispatching logic. Based on the USR, the policy selects a_t at each step. Once a_t is chosen, the environment deterministically transitions to a new state s_{t+1} by updating the graph topology and node- and edge-level features. Completed operations and machines with no remaining processable operations are removed along with their edges to prune historical information irrelevant to future decisions and reduce the graph size. For the selected action, EST_{ijm}^{t} and EFT_{ijm}^{t} become the actual start and finish times, namely S_{ij} and C_{ij}.

4.2 Unified Neural Solver

Model Overview. To process the graph-structured state, we develop a modular architecture with a dedicated Transformer block for each graph. All blocks follow a Pre-Norm Transformer-style architecture [40, 45] (see Appendix B.3 for details), while each block applies a Multi-Head Attention (MHA) mechanism customized to the semantics of its graph. We refer to the blocks for $\mathcal{G}_{M2O}$ and $\mathcal{G}_{O2M}$ as **Cross-Attention Blocks (CABs)**, which model bipartite relationships between operations and machines with attributed edges (O-M pairs), and those for $\mathcal{G}_{O2O}$ and $\mathcal{G}_{M2M}$ as **Self-Attention Blocks (SABs)**, which capture intra-type dependencies. We first project the raw operation, machine, and O-M pair features into a common d-dimensional space (for readability, we omit the step index t): $\boldsymbol{h}_{O_{ij}}^{(0)} = \boldsymbol{W}_O \boldsymbol{f}_{O_{ij}}, \boldsymbol{h}_{M_m}^{(0)} = \boldsymbol{W}_M \boldsymbol{f}_{M_m}, \boldsymbol{h}_{e_{ijm}}^{(0)} = \boldsymbol{W}_e \boldsymbol{f}_{e_{ijm}}$, where $\boldsymbol{W}_O \in \mathbb{R}^{d\times 2}$, $\boldsymbol{W}_M \in \mathbb{R}^{d\times 4}$, and $\boldsymbol{W}_e \in \mathbb{R}^{d\times 3}$ are learnable parameters. These embeddings are then passed through L stacked layers, each applying CABs followed by SABs. We deliberately place the CABs before the SABs, allowing unified constraint information encoded in O-M pair features to condition the embeddings before subsequent SABs refine the enriched representations (see Appendix B.4 for analysis). Finally, a **Decision-Making Head (DMH)** outputs dispatching decisions based on the resulting embeddings. The overall framework of SCHEDFORMER is illustrated in Figure 4.

Cross-Attention Blocks for $\mathcal{G}_{M2O}$ and $\mathcal{G}_{O2M}$. As a key component of the USR, O-M pair features tell the policy how the active constraints bear on each action. Using this local information to make

globally informed decisions thus depends on how effectively CABs incorporate these features into representation learning. To this end, we design an Evolving-Edge Multi-Head Cross-Attention (EE-MHCA) mechanism that adaptively aggregates information from compatible machines (or operations) into each operation (or machine) embedding while directly conditioning the attention computation on the corresponding edge information. We describe the computation for $\mathcal{G}_{M2O}$ below; $\mathcal{G}_{O2M}$ follows the same procedure in the reverse direction, with different query $\boldsymbol{q}$, key $\boldsymbol{k}$, and value $\boldsymbol{v}$ sources. At each layer l, we first update the edge embeddings via residual fusion to enhance their expressive capacity (for readability, we omit the graph-type index from the embedding and parameter notation):

$$\dot{\boldsymbol{h}}_{O_{ij}}^{(l-1)} = \text{Norm}(\boldsymbol{h}_{O_{ij}}^{(l-1)}),\ \dot{\boldsymbol{h}}_{M_m}^{(l-1)} = \text{Norm}(\boldsymbol{h}_{M_m}^{(l-1)}),\ \dot{\boldsymbol{h}}_{e_{ijm}}^{(l-1)} = \text{Norm}(\boldsymbol{h}_{e_{ijm}}^{(l-1)}), \tag{2}$$

$$\boldsymbol{h}_{e_{ijm}}^{(l)} = \boldsymbol{h}_{e_{ijm}}^{(l-1)} + \text{SiLU}(\boldsymbol{W}_1^{(l)}\dot{\boldsymbol{h}}_{O_{ij}}^{(l-1)} + \boldsymbol{W}_2^{(l)}\dot{\boldsymbol{h}}_{e_{ijm}}^{(l-1)} + \boldsymbol{W}_3^{(l)}\dot{\boldsymbol{h}}_{M_m}^{(l-1)}), \tag{3}$$

where Norm denotes root mean square normalization [46], $\boldsymbol{W}_1^{(l)}$, $\boldsymbol{W}_2^{(l)}$, and $\boldsymbol{W}_3^{(l)} \in \mathbb{R}^{d\times d}$ are learnable model weights, and SiLU indicates the Sigmoid Linear Unit activation function. Note that the normalized node embeddings in Equation (2) and the edge embeddings updated by Equation (3) are shared between the two CABs. We then inject $\boldsymbol{h}_{e_{ijm}}^{(l)}$ into the queries, keys, and values via additive mixing, allowing refined pairwise context to directly shape both the attention weights and the aggregated representation, as described in Equations (4)–(6):

$$\begin{gathered}\boldsymbol{q}_{ij}^{(l)h} = \boldsymbol{W}_q^{(l)h}\dot{\boldsymbol{h}}_{O_{ij}}^{(l-1)},\ \boldsymbol{k}_m^{(l)h} = \boldsymbol{W}_k^{(l)h}\dot{\boldsymbol{h}}_{M_m}^{(l-1)},\ \boldsymbol{v}_m^{(l)h} = \boldsymbol{W}_v^{(l)h}\dot{\boldsymbol{h}}_{M_m}^{(l-1)},\\ \boldsymbol{q}_{ijm}^{(l)h} = \boldsymbol{W}_{qe}^{(l)h}\text{Norm}(\boldsymbol{h}_{e_{ijm}}^{(l)}),\ \boldsymbol{k}_{ijm}^{(l)h} = \boldsymbol{W}_{ke}^{(l)h}\text{Norm}(\boldsymbol{h}_{e_{ijm}}^{(l)}),\ \boldsymbol{v}_{ijm}^{(l)h} = \boldsymbol{W}_{ve}^{(l)h}\text{Norm}(\boldsymbol{h}_{e_{ijm}}^{(l)}).\end{gathered} \tag{4}$$

$$a_{ijm}^{(l)h} = \text{Softmax}_{M_m\in\mathcal{M}_{ij}}\left(\frac{\langle \boldsymbol{q}_{ij}^{(l)h} + \boldsymbol{q}_{ijm}^{(l)h}, \boldsymbol{k}_m^{(l)h} + \boldsymbol{k}_{ijm}^{(l)h}\rangle}{\sqrt{d_h}}\right), \tag{5}$$

$$\hat{\boldsymbol{h}}_{O_{ij}}^{(l)} = \boldsymbol{h}_{O_{ij}}^{(l-1)} + \boldsymbol{W}_f^{(l)}\text{Concat}(\hat{\boldsymbol{h}}_{O_{ij}}^{(l)1}, \ldots, \hat{\boldsymbol{h}}_{O_{ij}}^{(l)H}),\ \text{where}\ \hat{\boldsymbol{h}}_{O_{ij}}^{(l)h} = \sum_{M_m\in\mathcal{M}_{ij}} a_{ijm}^{(l)h}(\boldsymbol{v}_m^{(l)h} + \boldsymbol{v}_{ijm}^{(l)h}). \tag{6}$$

Here, $\boldsymbol{W}_q^{(l)h}, \boldsymbol{W}_k^{(l)h}, \boldsymbol{W}_v^{(l)h}, \boldsymbol{W}_{qe}^{(l)h}, \boldsymbol{W}_{ke}^{(l)h}, \boldsymbol{W}_{ve}^{(l)h} \in \mathbb{R}^{d_h\times d}$, and $\boldsymbol{W}_f^{(l)} \in \mathbb{R}^{d\times d}$ are trainable projection parameters, $\langle\cdot,\cdot\rangle$ denotes the dot product, H represents the number of attention heads, and $d_h = d/H$ is the dimension of each head h. Finally, we apply nonlinear transformations to the updated embeddings with a residual connection: $\bar{\boldsymbol{h}}_{O_{ij}}^{(l)} = \hat{\boldsymbol{h}}_{O_{ij}}^{(l)} + \text{SwiGLU}(\text{Norm}(\hat{\boldsymbol{h}}_{O_{ij}}^{(l)}))$, where SwiGLU denotes a feed-forward network with SiLU-based gating [47].

Self-Attention Blocks for $\mathcal{G}_{O2O}$ and $\mathcal{G}_{M2M}$. We next feed the conditioned operation embeddings $\bar{\boldsymbol{h}}_{O_{ij}}^{(l)}$ and machine embeddings $\bar{\boldsymbol{h}}_{M_m}^{(l)}$ (the outputs of the CABs) into the SABs for $\mathcal{G}_{O2O}$ and $\mathcal{G}_{M2M}$, respectively. Each SAB employs a standard Multi-Head Self-Attention (MHSA) mechanism [32] with graph-specific modifications. For $\mathcal{G}_{O2O}$, we integrate Rotary Position Embedding (RoPE) [31, 48] into MHSA to capture sequential dependencies among operations within the same job. For $\mathcal{G}_{M2M}$, we do not use positional encoding, as there is no inherent ordering among machines. Details of the MHSA procedure and MHSA with RoPE ($\overrightarrow{\text{MHSA}}$) are provided in Appendix B.5. The outputs of the l-th layer are then computed as follows:

$$\boldsymbol{h}_{O_{ij}}^{(l)} = \tilde{\boldsymbol{h}}_{O_{ij}}^{(l)} + \text{SwiGLU}(\text{Norm}(\tilde{\boldsymbol{h}}_{O_{ij}}^{(l)})),\ \tilde{\boldsymbol{h}}_{O_{ij}}^{(l)} = \bar{\boldsymbol{h}}_{O_{ij}}^{(l)} + \overrightarrow{\text{MHSA}}(\text{Norm}(\bar{\boldsymbol{h}}_{O_{ij}}^{(l)}), \text{Norm}(\bar{\boldsymbol{h}}_{\mathcal{O}_j}^{(l)})), \tag{7}$$

$$\boldsymbol{h}_{M_m}^{(l)} = \tilde{\boldsymbol{h}}_{M_m}^{(l)} + \text{SwiGLU}(\text{Norm}(\tilde{\boldsymbol{h}}_{M_m}^{(l)})),\ \tilde{\boldsymbol{h}}_{M_m}^{(l)} = \bar{\boldsymbol{h}}_{M_m}^{(l)} + \text{MHSA}(\text{Norm}(\bar{\boldsymbol{h}}_{M_m}^{(l)}), \text{Norm}(\bar{\boldsymbol{h}}_{\mathcal{M}}^{(l)})), \tag{8}$$

where $\mathcal{O}_j$ denotes the set of uncompleted operations in job J_j.

Decision-Making Head for Action Selection. The DMH takes the deep embeddings from the final Transformer layer and produces a probability distribution over candidate actions. Specifically, we implement the DMH as a Multi-Layer Perceptron (MLP) that computes a scalar score for each candidate action, $u(a_t|s_t, \boldsymbol{x}) = \text{MLP}(\text{Concat}(\text{Norm}(\boldsymbol{h}_{O_{ij}}^{(L)}), \text{Norm}(\boldsymbol{h}_{M_m}^{(L)}), \boldsymbol{f}_{e_{ijm}}^t)), \forall a_t \in \mathcal{A}_t$. These scores are then normalized via softmax to obtain the action probabilities: $\pi_\theta(a_t|s_t, \boldsymbol{x}) = \frac{\exp(u(a_t|s_t,\boldsymbol{x}))}{\sum_{a'_t\in\mathcal{A}_t}\exp(u(a'_t|s_t,\boldsymbol{x}))}$. The action is selected (greedily or via sampling) from this distribution.

4.3 Training Algorithm

To train the unified neural solver, we use ASIL [49]. Specifically, for a given training instance $\boldsymbol{x}$, we sample a group of G solutions from the policy and select the one with the lowest makespan as the pseudo-label $\boldsymbol{\tau}^* = (a_1^*, \ldots, a_{|\mathcal{O}|}^*) = \arg\min_{\boldsymbol{\tau}^g \in \{\boldsymbol{\tau}^1, \ldots, \boldsymbol{\tau}^G\}} C_{\max}(\boldsymbol{\tau}^g, \boldsymbol{x})$. The policy then learns to imitate this pseudo-label, with the cross-entropy loss adaptively weighted by the standardized advantage $A(\boldsymbol{x})$ that reflects the quality of the pseudo-label:

$$\begin{aligned} \mathcal{L}_{\text{ASIL}} &= -A(\boldsymbol{x}) \frac{1}{|\mathcal{O}|} \sum_{t=1}^{|\mathcal{O}|} \log \pi_\theta(a_t^* \mid s_t^*, \boldsymbol{x}), \\ \text{where } A(\boldsymbol{x}) &= \frac{\text{mean}(\{C_{\max}(\boldsymbol{\tau}^1, \boldsymbol{x}), \ldots, C_{\max}(\boldsymbol{\tau}^G, \boldsymbol{x})\}) - C_{\max}(\boldsymbol{\tau}^*, \boldsymbol{x})}{\text{std}(\{C_{\max}(\boldsymbol{\tau}^1, \boldsymbol{x}), \ldots, C_{\max}(\boldsymbol{\tau}^G, \boldsymbol{x})\}) + \epsilon}. \end{aligned} \tag{9}$$

Here, s_t^* is the state at step t of $\boldsymbol{\tau}^*$, and $\epsilon > 0$ ensures numerical stability. ASIL is well suited to multi-task learning, as it requires neither expert labels nor variant-specific reward design, and its gradients are relatively insensitive to objective scales that differ widely across variants, mitigating potential bias toward specific variants. Further rationale for using ASIL is detailed in Appendix B.6.

5 Experiments

In this section, we empirically evaluate SCHEDFORMER as a generalist solver across all 16 FJSSP variants. Our experiments aim to address four research questions: **(RQ1)** How does SCHEDFORMER compare with exact, heuristic, and neural baselines, and how well does it generalize to unseen variants, as well as to larger instances and shifted data distributions? **(RQ2)** How does it perform on well-established FJSSP benchmarks? **(RQ3)** Can it transfer to other scheduling problems without retraining? **(RQ4)** What does each proposed component contribute to its overall performance? Our code is publicly available at `https://github.com/Ingukchoi/SchedFormer`.

5.1 Experimental Settings

Data Generation. We first generate classic FJSSP instances using the well-established SD_2 procedure [30]. This distribution is widely regarded as challenging due to its high machine flexibility and large variability in processing times. Based on these instances, we produce datasets for all 16 FJSSP variants. The detailed data generation process is described in Appendix C.1.

Model Settings. Our unified neural solver consists of $L = 2$ layers, each comprising four dedicated blocks, as outlined in Section 4.2. All blocks employ MHA with $H = 8$ heads and an embedding dimension of $d = 128$, following common practice. For the DMH, we use an MLP with two hidden layers of dimension 256, SiLU activations, and a single-unit output layer.

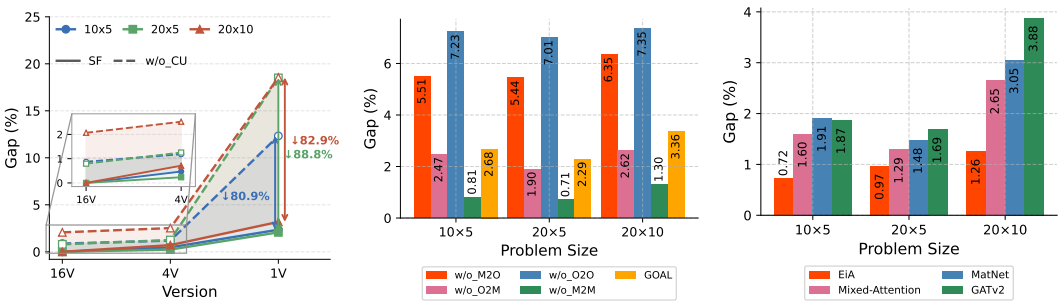
Training and Inference. We train three versions of SCHEDFORMER (SF) and evaluate them without further fine-tuning. **SF (16V)**, our main model, is trained on all 16 FJSSP variants. **SF (4V)** is trained on the four single-constraint variants (FJSSP-R, FJSSP-S, FJSSP-M, and FJSSP-T), while **SF (1V)** is trained only on the classic FJSSP. The latter two models are used to assess *cross-variant generalization*. To evaluate *cross-size generalization*, all models are trained only on 10×5 instances and tested across multiple problem sizes. For each variant and size, we generate 100 test instances using the same distribution as in training. We additionally evaluate *cross-distribution generalization* on test sets generated from distributions shifted from the training distribution. All models share the same training hyperparameters and use greedy decoding at test time unless otherwise stated. More detailed training settings and hyperparameters are presented in Appendix C.2.

Baseline Methods. We consider three types of baselines. **1) Exact solver**: CP-SAT solver from OR-Tools [50], using per-instance time limits of 60, 120, and 180 seconds (s) for $10 \times 5, 20 \times 5$, and 20×10 instances, respectively. **2) Classical PDRs**: First-In-First-Out (FIFO), Most Operations Remaining (MOR), and Most Work Remaining (MWKR). **3) Neural solvers**: To the best of our knowledge, there is no unified model for FJSSP variants. We therefore use representative neural FJSSP solvers (HGNN [29], DAN [30], and ReSched [31]) after adapting them to our multi-variant

Table 1: Experimental results on 16 FJSSP variants. Gray : Results of SCHEDFORMER. (↓): Lower is better. Gap: Relative gap in average makespan to SF (16V). Time: Average per-instance runtime. ×: OR-Tools fails to find a feasible solution for some instances. **Boldface** and underlining represent the best and second-best Gap among constructive methods, respectively. Due to space limitations, average makespan (MS) results for each method are reported in Appendix D.1.

	Method	10 × 5		20 × 5		20 × 10		Method	10 × 5		20 × 5		20 × 10	
		Gap (↓)	Time (↓)	Gap (↓)	Time (↓)	Gap (↓)	Time (↓)		Gap (↓)	Time (↓)	Gap (↓)	Time (↓)	Gap (↓)	Time (↓)
FJSSP	OR-Tools	-8.81%	60s	-3.95%	120s	-6.45%	180s	**FJSSP-SM** OR-Tools	5.82%	60s	×	×	×	×
	FIFO	18.34%	0.05s	22.48%	0.10s	24.09%	0.22s	FIFO	25.53%	0.05s	39.62%	0.10s	33.04%	0.23s
	MOR	17.33%	0.05s	23.09%	0.11s	21.38%	0.22s	MOR	23.40%	0.05s	39.20%	0.11s	31.73%	0.23s
	MWKR	15.02%	0.05s	19.78%	0.10s	20.12%	0.23s	MWKR	22.19%	0.05s	38.94%	0.11s	30.72%	0.23s
	HGNN	49.27%	0.15s	64.30%	0.28s	105.47%	0.59s	HGNN	40.76%	0.14s	59.56%	0.29s	74.18%	0.60s
	DAN	7.55%	0.23s	6.43%	0.45s	11.45%	0.92s	DAN	9.81%	0.23s	9.45%	0.46s	12.02%	0.93s
	ReSched	6.87%	0.27s	4.96%	0.53s	7.81%	1.06s	ReSched	7.04%	0.27s	5.16%	0.54s	6.58%	1.07s
	SF (16V)	0.00%	0.33s	**0.00%**	0.63s	0.00%	1.28s	SF (16V)	**0.00%**	0.33s	**0.00%**	0.63s	**0.00%**	1.28s
	SF (4V)	-1.00%	0.32s	0.23%	0.64s	**-1.00%**	1.29s	SF (4V)	0.82%	0.34s	0.72%	0.63s	1.79%	1.27s
	SF (1V)	**-1.25%**	0.33s	0.72%	0.63s	-0.22%	1.29s	SF (1V)	5.49%	0.33s	3.68%	0.62s	4.91%	1.28s
FJSSP-R	OR-Tools	-7.52%	60s	-4.45%	120s	-8.50%	180s	**FJSSP-ST** OR-Tools	-3.68%	60s	×	×	×	×
	FIFO	18.07%	0.04s	21.97%	0.10s	21.09%	0.23s	FIFO	22.71%	0.05s	39.63%	0.10s	26.22%	0.23s
	MOR	16.01%	0.05s	20.30%	0.10s	17.98%	0.23s	MOR	21.71%	0.05s	41.65%	0.11s	24.58%	0.23s
	MWKR	15.21%	0.05s	18.16%	0.11s	18.21%	0.24s	MWKR	21.91%	0.05s	39.39%	0.11s	23.36%	0.24s
	HGNN	47.03%	0.14s	64.57%	0.29s	91.28%	0.59s	HGNN	42.37%	0.14s	67.70%	0.29s	75.64%	0.59s
	DAN	7.65%	0.23s	7.08%	0.45s	9.73%	0.90s	DAN	10.38%	0.23s	8.78%	0.45s	10.99%	0.92s
	ReSched	7.27%	0.27s	5.55%	0.54s	6.80%	1.08s	ReSched	9.15%	0.27s	5.13%	0.55s	7.35%	1.06s
	SF (16V)	0.00%	0.32s	**0.00%**	0.62s	0.00%	1.29s	SF (16V)	**0.00%**	0.33s	**0.00%**	0.63s	**0.00%**	1.27s
	SF (4V)	**-0.02%**	0.33s	0.18%	0.63s	**-0.30%**	1.27s	SF (4V)	1.40%	0.34s	0.04%	0.64s	0.69%	1.27s
	SF (1V)	0.18%	0.33s	0.31%	0.63s	2.46%	1.28s	SF (1V)	3.07%	0.33s	1.41%	0.62s	3.02%	1.29s
FJSSP-S	OR-Tools	-2.56%	60s	×	×	×	×	**FJSSP-MT** OR-Tools	-1.43%	60s	×	×	×	×
	FIFO	25.55%	0.05s	41.98%	0.10s	35.40%	0.22s	FIFO	16.94%	0.05s	22.15%	0.11s	20.41%	0.22s
	MOR	24.40%	0.05s	43.88%	0.11s	34.66%	0.22s	MOR	15.14%	0.05s	23.18%	0.11s	18.09%	0.22s
	MWKR	24.27%	0.05s	41.52%	0.11s	33.41%	0.23s	MWKR	13.63%	0.05s	21.46%	0.11s	17.58%	0.23s
	HGNN	47.86%	0.15s	69.63%	0.29s	91.32%	0.60s	HGNN	37.69%	0.15s	52.80%	0.29s	76.00%	0.60s
	DAN	8.87%	0.24s	9.31%	0.45s	13.43%	0.92s	DAN	7.69%	0.23s	6.80%	0.46s	9.62%	0.92s
	ReSched	7.43%	0.28s	3.31%	0.55s	6.87%	1.06s	ReSched	8.68%	0.27s	7.92%	0.54s	9.87%	1.06s
	SF (16V)	**0.00%**	0.33s	0.00%	0.65s	**0.00%**	1.28s	SF (16V)	0.00%	0.33s	**0.00%**	0.63s	**0.00%**	1.27s
	SF (4V)	0.63%	0.32s	**-0.05%**	0.63s	1.39%	1.28s	SF (4V)	**-0.16%**	0.33s	0.34%	0.63s	0.08%	1.28s
	SF (1V)	2.66%	0.34s	0.87%	0.63s	3.82%	1.28s	SF (1V)	3.06%	0.33s	3.28%	0.63s	3.63%	1.29s
FJSSP-M	OR-Tools	0.61%	60s	×	×	×	×	**FJSSP-RSM** OR-Tools	6.63%	60s	×	×	×	×
	FIFO	21.57%	0.05s	22.17%	0.11s	24.50%	0.23s	FIFO	23.94%	0.05s	36.58%	0.10s	31.16%	0.23s
	MOR	19.07%	0.05s	22.97%	0.11s	22.24%	0.22s	MOR	22.67%	0.05s	36.58%	0.12s	29.28%	0.22s
	MWKR	17.13%	0.05s	20.42%	0.11s	22.03%	0.23s	MWKR	20.76%	0.05s	36.57%	0.11s	28.71%	0.23s
	HGNN	41.31%	0.15s	50.17%	0.29s	80.49%	0.62s	HGNN	40.12%	0.15s	59.48%	0.30s	70.92%	0.61s
	DAN	9.70%	0.23s	6.16%	0.46s	11.04%	0.92s	DAN	8.60%	0.22s	9.12%	0.44s	11.88%	0.94s
	ReSched	8.01%	0.27s	5.71%	0.54s	8.53%	1.07s	ReSched	6.59%	0.27s	4.25%	0.56s	6.27%	1.05s
	SF (16V)	**0.00%**	0.33s	0.00%	0.63s	0.00%	1.28s	SF (16V)	**0.00%**	0.33s	**0.00%**	0.63s	**0.00%**	1.28s
	SF (4V)	0.91%	0.33s	**-0.09%**	0.63s	**-0.57%**	1.29s	SF (4V)	1.31%	0.33s	0.59%	0.64s	1.35%	1.27s
	SF (1V)	2.96%	0.33s	2.94%	0.62s	3.87%	1.28s	SF (1V)	4.51%	0.32s	3.36%	0.63s	3.99%	1.26s
FJSSP-T	OR-Tools	-10.08%	60s	-3.88%	120s	-6.76%	180s	**FJSSP-RST** OR-Tools	-4.43%	60s	×	×	×	×
	FIFO	16.63%	0.05s	22.60%	0.10s	16.77%	0.22s	FIFO	20.06%	0.05s	38.21%	0.11s	24.86%	0.22s
	MOR	13.51%	0.05s	23.09%	0.11s	15.38%	0.22s	MOR	18.09%	0.05s	36.62%	0.10s	22.16%	0.22s
	MWKR	12.94%	0.05s	21.15%	0.11s	14.90%	0.23s	MWKR	17.41%	0.05s	37.08%	0.11s	21.63%	0.22s
	HGNN	42.60%	0.15s	66.86%	0.29s	89.31%	0.60s	HGNN	37.26%	0.15s	67.62%	0.29s	69.99%	0.60s
	DAN	5.34%	0.23s	7.09%	0.45s	8.56%	0.92s	DAN	7.80%	0.23s	8.83%	0.45s	11.02%	0.92s
	ReSched	7.41%	0.27s	10.04%	0.54s	9.12%	1.06s	ReSched	6.44%	0.27s	5.01%	0.54s	8.21%	1.06s
	SF (16V)	0.00%	0.34s	0.00%	0.63s	0.00%	1.28s	SF (16V)	**0.00%**	0.33s	**0.00%**	0.62s	**0.00%**	1.27s
	SF (4V)	**-0.35%**	0.33s	**-0.06%**	0.63s	**-0.18%**	1.28s	SF (4V)	0.16%	0.33s	0.12%	0.63s	0.98%	1.28s
	SF (1V)	-0.02%	0.33s	0.23%	0.64s	1.35%	1.28s	SF (1V)	2.31%	0.33s	1.78%	0.64s	4.03%	1.27s
FJSSP-RT	OR-Tools	-8.66%	60s	-4.63%	120s	-9.53%	180s	**FJSSP-RMT** OR-Tools	-2.97%	60s	×	×	×	×
	FIFO	14.81%	0.05s	23.41%	0.10s	15.12%	0.21s	FIFO	16.55%	0.05s	22.39%	0.10s	20.19%	0.22s
	MOR	12.46%	0.05s	22.40%	0.10s	11.38%	0.23s	MOR	13.23%	0.05s	23.28%	0.11s	14.97%	0.22s
	MWKR	12.60%	0.05s	20.10%	0.11s	7.18%	0.23s	MWKR	13.89%	0.05s	20.79%	0.11s	14.21%	0.22s
	HGNN	37.81%	0.14s	64.85%	0.29s	69.80%	0.59s	HGNN	34.46%	0.15s	52.86%	0.29s	65.19%	0.59s
	DAN	5.76%	0.24s	7.79%	0.46s	5.59%	0.90s	DAN	7.91%	0.23s	8.20%	0.45s	8.45%	0.91s
	ReSched	5.61%	0.28s	8.22%	0.54s	4.35%	1.06s	ReSched	7.40%	0.27s	7.82%	0.55s	7.67%	1.06s
	SF (16V)	**0.00%**	0.33s	**0.00%**	0.63s	0.00%	1.28s	SF (16V)	**0.00%**	0.34s	**0.00%**	0.63s	**0.00%**	1.28s
	SF (4V)	0.14%	0.33s	0.16%	0.62s	**-0.06%**	1.27s	SF (4V)	0.19%	0.33s	0.10%	0.63s	1.63%	1.27s
	SF (1V)	0.24%	0.32s	0.07%	0.63s	2.05%	1.27s	SF (1V)	2.17%	0.33s	3.70%	0.63s	2.61%	1.27s
FJSSP-RS	OR-Tools	-2.93%	60s	×	×	×	×	**FJSSP-SMT** OR-Tools	2.50%	60s	×	×	×	×
	FIFO	23.29%	0.05s	41.48%	0.11s	31.05%	0.23s	FIFO	21.41%	0.05s	37.59%	0.11s	27.20%	0.22s
	MOR	22.72%	0.04s	40.34%	0.11s	29.20%	0.23s	MOR	20.88%	0.05s	37.10%	0.10s	26.39%	0.22s
	MWKR	21.27%	0.05s	40.06%	0.11s	29.40%	0.23s	MWKR	19.15%	0.04s	37.44%	0.11s	25.73%	0.23s
	HGNN	44.51%	0.15s	70.18%	0.30s	82.78%	0.59s	HGNN	35.78%	0.14s	57.50%	0.28s	66.32%	0.59s
	DAN	8.71%	0.23s	9.39%	0.45s	11.69%	0.90s	DAN	9.52%	0.23s	9.13%	0.45s	10.79%	0.92s
	ReSched	5.56%	0.27s	4.24%	0.54s	5.55%	1.06s	ReSched	7.70%	0.27s	5.56%	0.54s	7.44%	1.06s
	SF (16V)	**0.00%**	0.33s	**0.00%**	0.63s	0.00%	1.28s	SF (16V)	**0.00%**	0.33s	**0.00%**	0.63s	**0.00%**	1.28s
	SF (4V)	0.18%	0.32s	0.30%	0.63s	**-0.66%**	1.27s	SF (4V)	1.06%	0.33s	0.56%	0.64s	2.93%	1.28s
	SF (1V)	1.70%	0.33s	1.29%	0.64s	1.90%	1.28s	SF (1V)	5.03%	0.32s	3.11%	0.63s	5.10%	1.29s
FJSSP-RM	OR-Tools	0.13%	60s	×	×	×	×	**FJSSP-RSMT** OR-Tools	4.61%	60s	×	×	×	×
	FIFO	18.75%	0.05s	21.68%	0.10s	25.40%	0.22s	FIFO	20.80%	0.05s	35.03%	0.11s	26.60%	0.23s
	MOR	15.83%	0.05s	19.91%	0.11s	21.10%	0.23s	MOR	19.42%	0.05s	34.91%	0.11s	23.02%	0.22s
	MWKR	15.16%	0.05s	19.32%	0.11s	20.59%	0.22s	MWKR	18.14%	0.05s	35.01%	0.10s	22.37%	0.23s
	HGNN	37.56%	0.14s	50.72%	0.29s	75.98%	0.60s	HGNN	35.04%	0.15s	57.07%	0.29s	62.09%	0.58s
	DAN	8.42%	0.23s	6.40%	0.44s	11.65%	0.89s	DAN	8.96%	0.23s	8.71%	0.46s	10.73%	0.91s
	ReSched	6.20%	0.27s	5.73%	0.54s	8.41%	1.07s	ReSched	7.49%	0.27s	5.28%	0.53s	7.54%	1.07s
	SF (16V)	**0.00%**	0.33s	0.00%	0.62s	**0.00%**	1.28s	SF (16V)	**0.00%**	0.33s	**0.00%**	0.64s	**0.00%**	1.28s
	SF (4V)	0.07%	0.33s	**-0.17%**	0.63s	1.20%	1.27s	SF (4V)	2.57%	0.33s	0.84%	0.64s	2.10%	1.28s
	SF (1V)	2.37%	0.33s	2.47%	0.63s	4.39%	1.27s	SF (1V)	3.32%	0.34s	3.91%	0.63s	4.04%	1.28s

setting. Specifically, we provide each solver with the same state features used by SCHEDFORMER, including EST and EFT, while preserving its original architecture. All neural baselines are trained and evaluated under the same protocol as SF (16V). Baseline details are provided in Appendix C.3.



(a) Unification ablation **(b)** Block ablation **(c)** MHCA comparison

Figure 5: **Ablation study results.** Gaps (%) are averaged over 16 variants and reported relative to SF (16V). **(a):** 16V, 4V, and 1V denote the number of training variants, matching the corresponding SF settings. Arrows show the relative gap reduction of SF over w/o_CU at 1V. **(b):** w/o_M2O, w/o_O2M, w/o_O2O, and w/o_M2M remove the corresponding graph-specific block. GOAL shares parameters across the four blocks. **(c):** Alternative mechanisms for incorporating edge features into attention. For **(b)** and **(c)**, all ablated models follow the same training setting as SF (16V).

5.2 Experimental Results

(RQ1) Main Results. Table 1 summarizes the main results. SCHEDFORMER consistently outperforms all PDRs and neural baselines and surpasses OR-Tools in 30 of 48 cases, including 24 cases in which OR-Tools fails to find feasible solutions for some instances within the time limit, while being orders of magnitude faster. Notably, SCHEDFORMER shows strong cross-variant generalization. Against the best neural baseline on each dataset, SF (16V) and SF (4V) achieve statistically significant improvements on all 48 datasets ($p < 0.001$, two-sided Wilcoxon signed-rank test [51]). SF (1V) also significantly outperforms the best neural baseline on all datasets ($p < 0.001$ on 38, $0.001 \leq p < 0.01$ on five, and $0.01 \leq p < 0.05$ on the remaining five). Since most test variants lie outside the training variant space of SF (4V) and SF (1V), their results are largely zero-shot. Moreover, because the neural baselines share the same state features as SCHEDFORMER, the across-the-board margin suggests the contribution of our architectural design. This advantage also persists under distribution shift, demonstrating the robustness of SCHEDFORMER (see Appendix D.2). Additional results on the comparison with a metaheuristic, scalability to 40×10, 50×20, and 100×20 instances, and Best-of-N sampling performance are reported in Appendices D.3–D.5.

(RQ2) Benchmark Results. We next evaluate SCHEDFORMER on well-established FJSSP benchmarks. On the Hurink datasets (edata, rdata, and vdata) [52], which include larger instance sizes and differ from our training data in distribution, we compare against nine recent neural solvers, all designed and trained for the classic FJSSP. Although SCHEDFORMER is trained as a generalist, it reduces the gap to the best-known solutions by up to 39.1% relative to the strongest neural baseline. We further test scalability on the Behnke benchmark [53], which includes realistic manufacturing settings and instances up to 100×60. Here, SCHEDFORMER again achieves the best overall performance, confirming strong generalization to substantially larger instances. Details and results for each benchmark are provided in Appendices D.6 (Hurink) and D.7 (Behnke).

(RQ3) Generalization to Other Scheduling Problems. Since the FJSSP serves as a generic framework for multiple machine environments, SCHEDFORMER can be applied to other scheduling problems. As a showcase, we transfer it zero-shot to the Job Shop Scheduling Problem (JSSP), where each operation has a fixed machine assignment, and the Flexible Flow Shop Scheduling Problem (FFSP), where jobs follow a common sequence of processing stages. Without any model modification, SCHEDFORMER matches or surpasses specialized SOTA neural solvers on both problems, demonstrating its versatility as a general-purpose scheduler. Details of the benchmarks, baselines, and results for both problems are provided in Appendices D.8 (JSSP) and D.9 (FFSP).

5.3 (RQ4) Ablation Study

Effect of Constraint Unification. Here, we assess the effect of Constraint Unification (CU) by comparing SCHEDFORMER with three ablated models (w/o_CU (16V), w/o_CU (4V), and

w/o_CU (1V)), which encode the constraints separately rather than mapping them into a unified form, similar to existing combination-based approaches (see Appendix D.10 for feature definitions). As shown in Figure 5(a), all ablated models underperform our models, most severely in the 1V case, as the policy cannot interpret the effects of constraints it has never observed during training. In contrast, our unified features give the policy a common interface for interpreting the effects of new constraints, so its learned knowledge can transfer to unseen variants. These results highlight the importance of constraint unification for cross-variant generalization and, together with the results in Table 1, show that combining it with our architecture yields a powerful multi-task solver.

Effect of Graph-Specific Blocks. To validate our relation-specific graph design, we consider five ablated models of SCHEDFORMER. Four ablated models remove one relation-specific block at a time, while the fifth shares parameters across the four blocks using a 9-layer Transformer following GOAL [43]. Figure 5(b) shows that removing any single block degrades performance, indicating that the four relations capture complementary structural views. The parameter-sharing model also falls short of SF (16V) despite its greater depth, supporting the benefit of relation-specific parameters.

Comparison of Cross-Attention Mechanisms. To examine the contribution of our EE-MHCA, we replace it in the CABs with four representative alternatives used in NCO, each incorporating raw edge features through a different mechanism: EiA [31], Mixed-Attention [43], MatNet [54], and GATv2 [25]. The rest of the architecture is kept fixed. As illustrated in Figure 5(c), EE-MHCA achieves the lowest average gap at each problem size, confirming that our way of injecting evolving edge embeddings into the attention computation is a sensible design choice.

6 Conclusion, Limitations, and Future Work

Conclusion. This paper takes a step toward unified learning-based scheduling. We have presented SCHEDFORMER, built on the insight that many constraints can be interpreted as delays in when operations start and finish. The USR encodes these delays together with invariant operation-machine relations, and graph-specific Transformer blocks process the resulting representation in a unified neural solver trained with ASIL. As a result, SCHEDFORMER handles all variants considered and generalizes zero-shot to those unseen during training, even outperforming SOTA neural solvers trained on every variant. It also transfers to the JSSP and the FFSP without retraining, matching or surpassing neural solvers specialized for each problem. Finally, comprehensive ablations confirm that each component of our design contributes to its performance and generalization. We believe SCHEDFORMER can reduce the engineering effort required to deploy learning-based schedulers in real-world systems, where requirements vary across industries and sites and evolve over time.

Limitations and Future Work. Despite this promise, SCHEDFORMER has several limitations. First, the framework is restricted to constraints that can be expressed as temporal effects. Although this class already covers a wide range of practical scheduling constraints, it does not capture every type of constraint. Second, our evaluation relies on synthetic instances and public benchmarks, as real-world scheduling data are often proprietary and rarely released publicly. It also focuses on static settings, whereas real-world environments inevitably involve uncertainties such as stochastic processing times and unexpected disruptions. Third, our feature design is tailored to makespan minimization. Other objectives, such as total tardiness, may benefit from additional features (e.g., the slack between due dates and EFT), which could be added to the USR as shared node or edge attributes, thereby preserving a single representation for all variants. Important directions toward foundation models for scheduling therefore include extending the framework to broader constraint types and objectives, as well as validating it on industrial data and under more realistic scenarios.

Acknowledgments and Disclosure of Funding

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Grant: RS-2024-00334171), in part by the Basic Science Research Program through the NRF funded by the Ministry of Education (Grant: RS-2026-25552971), and in part by the Ministry of Science and ICT (MSIT), Republic of Korea, through the National IT Industry Promotion Agency (NIPA) (Grant: RS-2026-25621649).

References

[1] Jose M Framinan and Rubén Ruiz. Architecture of manufacturing scheduling systems: Literature review and an integrated proposal. *European Journal of Operational Research*, 205(2):237–246, 2010.

[2] N Sarfaraj, Md LR Lingkon, and N Zahan. Applying flexible job shop scheduling in patients management to optimize processing time in hospitals. *International Journal of Research in Industrial Engineering*, 10(1):46–55, 2021.

[3] Amir Hosseini, Alena Otto, and Erwin Pesch. Scheduling in manufacturing with transportation: Classification and solution techniques. *European Journal of Operational Research*, 315(3):821–843, 2024.

[4] AR Arunarani, Dhanabalachandran Manjula, and Vijayan Sugumaran. Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems*, 91:407–415, 2019.

[5] 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 *Advances in Neural Information Processing Systems*, volume 33, pages 1621–1632, 2020.

[6] Xiehui Zhang and Guang-Yu Zhu. A literature review of reinforcement learning methods applied to job-shop scheduling problems. *Computers & Operations Research*, 175:106929, 2025.

[7] Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Lin Xie, and Jinkyoo Park. RL4CO: An extensive reinforcement learning for combinatorial optimization benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025.

[8] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, NY, USA, 4th edition, 2012.

[9] Dar-Li Yang, Chou-Jung Hsu, and Wen-Hung Kuo. A two-machine flowshop scheduling problem with a separated maintenance constraint. *Computers & Operations Research*, 35(3):876–883, 2008.

[10] Rubén Ruiz and Concepción Maroto. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research*, 169(3):781–800, 2006.

[11] Sihoon Moon, Sanghoon Lee, and Kyung-Joon Park. Learning-enabled flexible job-shop scheduling for scalable smart manufacturing. *Journal of Manufacturing Systems*, 77:356–367, 2024.

[12] In-Beom Park, Jaeseok Huh, Joongkyun Kim, and Jonghun Park. A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities. *IEEE Transactions on Automation Science and Engineering*, 17(3):1420–1431, 2020.

[13] Wenda Wang, Yi Zhang, Yong Wang, Ge Pan, and Yiping Feng. Hierarchical multi-agent deep reinforcement learning for dynamic flexible job-shop scheduling with transportation. *International Journal of Production Research*, pages 1–28, 2025.

[14] Kjell van Straaten, Robbert Reijnen, Zaharah Bukhsh, Yaoxin Wu, and Yingqian Zhang. Deep reinforcement learning based genetic framework for flexible job-shop scheduling under practical constraints. In *International Conference on Learning and Intelligent Optimization*, pages 183–198, 2025.

[15] Helio Yochihiro Fuchigami and Socorro Rangel. A survey of case studies in production scheduling: Analysis and perspectives. *Journal of Computational Science*, 25:425–436, 2018.

[16] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

[17] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[18] Kun Lei, Peng Guo, Wenchao Zhao, Yi Wang, Linmao Qian, Xiangyin Meng, and Liansheng Tang. A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem. *Expert Systems with Applications*, 205:117796, 2022.

[19] Imanol Echeverria, Maialen Murua, and Roberto Santana. Solving the flexible job-shop scheduling problem through an enhanced deep reinforcement learning approach. *arXiv preprint arXiv:2310.15706*, 2023.

[20] Peng Zhao, You Zhou, Di Wang, Zhiguang Cao, Yubin Xiao, Xuan Wu, Yuanshu Li, Hongjia Liu, Wei Du, Yuan Jiang, et al. Dual operation aggregation graph neural networks for solving flexible job-shop scheduling problem with reinforcement learning. In *Proceedings of the ACM on Web Conference 2025*, pages 4089–4100, 2025.

[21] Erdong Yuan, Liejun Wang, Shuli Cheng, Shiji Song, Wei Fan, and Yongming Li. Solving flexible job shop scheduling problems via deep reinforcement learning. *Expert Systems with Applications*, 245:123019, 2024.

[22] Dailin Huang, Hong Zhao, Jie Cao, Kangping Chen, and Lijun Zhang. Optimizing the flexible job shop scheduling problem via deep reinforcement learning with mean multichannel graph attention. *Applied Soft Computing*, 177:113128, 2025.

[23] Dailin Huang, Hong Zhao, Weiquan Tian, and Kangping Chen. A deep reinforcement learning method based on a multiexpert graph neural network for flexible job shop scheduling. *Computers & Industrial Engineering*, 200:110768, 2025.

[24] Yan Kang, Tianjing Li, Lei Zhao, Zhan Gao, Jie He, Zhuangzhuang Chen, and Bin Pu. Dual dynamic graph attention network driven deep reinforcement learning for flexible job-shop scheduling. *Knowledge-Based Systems*, page 115192, 2025.

[25] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

[26] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[27] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198, 2020.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19(2):1600–1610, 2023.

[30] Runqing Wang, Gang Wang, Jian Sun, Fang Deng, and Jie Chen. Flexible job shop scheduling via dual attention network-based reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 35(3):3091–3102, 2024.

[31] Xiangjie Xiao, Zhiguang Cao, Cong Zhang, and Wen Song. ReSched: Rethinking flexible job shop scheduling from a Transformer-based architecture with simplified states. In *International Conference on Learning Representations*, 2026.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[33] Yu Du, Junqing Li, Chengdong Li, and Peiyong Duan. A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):5695–5709, 2024.

[34] Binxiao Yan, Xinbao Liu, Shaojun Lu, Chaoming Hu, Xubiao Wang, and Zhiping Zhou. Deep reinforcement learning approach for a dynamic flexible job shop problem with sequence dependent setup times. *Computers & Industrial Engineering*, page 111310, 2025.

[35] Nada Ouahabi, Ahmed Chebak, Oulaid Kamach, and Mourad Zegrari. Dynamic production scheduling and maintenance planning under opportunistic grouping. *Computers & Industrial Engineering*, 199:110646, 2025.

[36] Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1898–1908, 2024.

[37] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

[38] Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Chi Xu. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *International Conference on Machine Learning*, volume 235, pages 61804–61824, 2024.

[39] Yong Liang Goh, Zhiguang Cao, Yining Ma, Jianan Zhou, Mohammed Haroon Dupty, and Wee Sun Lee. SHIELD: Multi-task multi-distribution vehicle routing solver with sparsity and hierarchy. In *International Conference on Machine Learning*, volume 267, pages 19679–19711, 2025.

[40] Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. RouteFinder: Towards foundation models for vehicle routing problems. *Transactions on Machine Learning Research*, 2025.

[41] Han Li, Fei Liu, Zhi Zheng, Yu Zhang, and Zhenkun Wang. CaDA: Cross-problem routing solver with constraint-aware dual-attention. In *International Conference on Machine Learning*, volume 267, pages 35438–35456, 2025.

[42] Zhuoyi Lin, Yaoxin Wu, Bangjian Zhou, Zhiguang Cao, Wen Song, Yingqian Zhang, and Senthilnath Jayavelu. Cross-problem learning for solving vehicle routing problems. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 6958–6966, 2024.

[43] Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. GOAL: A generalist combinatorial optimization agent learner. In *International Conference on Learning Representations*, 2025.

[44] Stéphane Dauzère-Pérès, Junwen Ding, Liji Shen, and Karim Tamssaouet. The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 314(2):409–432, 2024.

[45] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the Transformer architecture. In *International Conference on Machine Learning*, volume 119, pages 10524–10533, 2020.

[46] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[47] Noam Shazeer. GLU variants improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[48] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced Transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[49] Inguk Choi, Woo-Jin Shin, Sang-Hyun Cho, and Hyun-Jung Kim. Towards generalizable multi-policy optimization with self-evolution for job scheduling. In *Advances in Neural Information Processing Systems*, volume 38, pages 24569–24616, 2025.

[50] Laurent Perron and Frédéric Didier. CP-SAT. `https://developers.google.com/optimization/cp/cp_solver/`, 2026. OR-Tools v9.15.

[51] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics: Methodology and Distribution*, pages 196–202. Springer, 1992.

[52] Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4):205–215, 1994.

[53] Dennis Behnke and Martin Josef Geiger. Test instances for the flexible job shop scheduling problem with work centers. Arbeitspapier/research paper, Helmut-Schmidt-Universität, Lehrstuhl für Betriebswirtschaftslehre, insbes. Logistik-Management, 2012.

[54] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 34, pages 5138–5149, 2021.

[55] Xiaosong Hu, Le Xu, Xianke Lin, and Michael Pecht. Battery lifetime prognostics. *Joule*, 4(2):310–346, 2020.

[56] Woo-Jin Shin, Sang-Wook Lee, Jun-Ho Lee, Min-Ho Song, and Hyun-Jung Kim. Scheduling of steelmaking-continuous casting process by integrating deep neural networks with mixed integer programming. *International Journal of Production Research*, 63(11):4180–4201, 2025.

[57] Sang-Wook Lee and Hyun-Jung Kim. Single machine scheduling with variable maintenance in the battery manufacturing. *International Journal of Production Research*, 63(23):8859–8879, 2025.

[58] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[59] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[60] Scott J Mason, John W Fowler, and W Matthew Carlyle. A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling*, 5(3):247–262, 2002.

[61] Sebastian Knopp, Stéphane Dauzère-Pérès, and Claude Yugma. A batch-oblivious approach for complex job-shop scheduling problems. *European Journal of Operational Research*, 263(1):50–61, 2017.

[62] Rubén Ruiz and Thomas Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008.

[63] B Naderi, M Zandieh, and SMT Fatemi Ghomi. Scheduling job shop problems with sequence-dependent setup times. *International Journal of Production Research*, 47(21):5959–5976, 2009.

[64] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[65] José Fernando Gonçalves and Mauricio GC Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.

[66] S Mahdi Homayouni, Dalila BMM Fontes, and José F Gonçalves. A multistart biased random key genetic algorithm for the flexible job shop scheduling problem with transportation. *International Transactions in Operational Research*, 30(2):688–716, 2023.

[67] Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. INViT: A generalizable routing problem solver with invariant nested view Transformer. In *International Conference on Machine Learning*, volume 235, pages 12973–12992, 2024.

[68] Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[69] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

[70] S Lawrence. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. *GSIA, Carnegie Mellon University*, 1984.

[71] Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. In *Advances in Neural Information Processing Systems*, volume 37, pages 105528–105551, 2024.

[72] Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. ScheduleNet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021.

[73] 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 Thirty-Second International Joint Conference on Artificial Intelligence*, pages 5350–5358, 2023.

[74] André Hottung, Mridul Mahajan, and Kevin Tierney. PolyNet: Learning diverse solution strategies for neural combinatorial optimization. In *International Conference on Learning Representations*, 2025.

[75] Federico Berto, Chuanbo Hua, Laurin Luttmann, Jiwoo Son, Junyoung Park, Kyuree Ahn, Changhyun Kwon, Lin Xie, and Jinkyoo Park. PARCO: Parallel autoregressive models for multi-agent combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 38, pages 167130–167160, 2025.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state our main contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our work in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all implementation details in this paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Our code and dataset are publicly available.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide all details regarding the experiments in Section 5.1 and Appendix C.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report statistical significance.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix C provides details on computational resources.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have checked the paper according to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the broader impact of this work in Appendix E.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We clearly state the sources of the original code, data, and models used in this work.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Our code and dataset are publicly available.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing experiments or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing experiments or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

Appendix Contents

A Flexible Job Shop Scheduling Problem Variants 24

B SCHEDFORMER Details 25
- **B.1** EST and EFT Computation 25
- **B.2** Node and Edge Features 26
- **B.3** Pre-Norm Transformer 26
- **B.4** Solver Design Analysis 26
- **B.5** Multi-Head Self-Attention 27
- **B.6** Rationale for Using ASIL 28

C Experimental Details 29
- **C.1** Data Generation Procedure 29
- **C.2** Training Setup 30
- **C.3** Baseline Implementation 30

D Additional Experiments 31
- **D.1** Makespan Results 31
- **D.2** Generalization under Distribution Shifts 32
- **D.3** Comparison with a Metaheuristic 32
- **D.4** Scalability to Larger Instances 34
- **D.5** Sampling Performance 35
- **D.6** The Hurink Benchmark 35
- **D.7** The Behnke Benchmark 36
- **D.8** Cross-Problem Generalization to the JSSP 37
- **D.9** Cross-Problem Generalization to the FFSP 38
- **D.10** Node and Edge Features of Models without Constraint Unification 39

E Broader Impact 39

Towards a Unified Model for Flexible Job Shop Scheduling with Diverse Constraints (Appendix)

Table 2: 16 FJSSP variants.

FJSSP variants	Release time (R)	Setup time (S)	Machine maintenance (M)	Transportation time (T)
FJSSP				
FJSSP-R	✓			
FJSSP-S		✓		
FJSSP-M			✓	
FJSSP-T				✓
FJSSP-RT	✓			✓
FJSSP-RS	✓	✓		
FJSSP-RM	✓		✓	
FJSSP-SM		✓	✓	
FJSSP-ST		✓		✓
FJSSP-MT			✓	✓
FJSSP-RSM	✓	✓	✓	
FJSSP-RST	✓	✓		✓
FJSSP-RMT	✓		✓	✓
FJSSP-SMT		✓	✓	✓
FJSSP-RSMT	✓	✓	✓	✓

A Flexible Job Shop Scheduling Problem Variants

From the four operational constraints introduced in Section 3.1, we define 16 FJSSP variants by enumerating all possible constraint combinations, as summarized in Table 2. Many of these variants have not yet been addressed by neural methods. We detail each constraint below.

Release Time (R). This constraint reflects practical scenarios arising from material arrivals, upstream process completion, and order-release policies, in which some jobs are initially available while others arrive over time. Formally, it imposes $S_{1j} \geq RT_j$.

Sequence-Dependent Setup Time (S). This constraint represents a general setup model in which setup times depend on the sequence of operations on each machine, capturing preparatory activities such as tool changes, cleaning, and recalibration. Formally, if operation $O_{i'j'}$ is processed immediately after O_{ij} on machine M_m, its start time must satisfy $S_{i'j'} \geq AT_m^t + ST_{O_{ij}O_{i'j'}m}$.

Machine Maintenance (M). This constraint models preventive maintenance, where machines must be periodically serviced to ensure reliability and product quality. Unlike time-based formulations adopted in some prior studies, we define maintenance as a function of the cumulative number of processed operations, which better reflects practice in many manufacturing settings [55, 56, 57]. Formally, let N_m^t denote, at step t, the number of operations assigned to M_m since its last maintenance. Then, M_m becomes available to process the next operation at time $AT_m^t + \mathbb{I}\{N_m^t = MI_m\}MD_m$, where $\mathbb{I}\{\cdot\}$ is the indicator function.

Transportation Time (T). This constraint captures the time required to move a job between machines. Such transfers commonly arise in manufacturing environments with material handling

systems. Formally, if operation O_{ij} is processed on machine M_m and its successor $O_{(i+1)j}$ is assigned to machine $M_{m'}$, then the successor's start time must satisfy $S_{(i+1)j} \geq C_{ij} + TT_{mm'}$.

General Assumptions. All problem variants are subject to the following standard assumptions: 1) all machines are available at time zero; 2) preemption is not allowed, meaning that once an operation starts, it must be completed without interruption; and 3) each machine can process at most one operation at a time.

B SCHEDFORMER Details

Due to space limitations, some details of SCHEDFORMER are omitted from the main paper. This section provides comprehensive descriptions of the SCHEDFORMER components.

B.1 EST and EFT Computation

Let $m_{(i-1)j}$ denote the index of the machine assigned to $O_{(i-1)j}$ for $i > 1$. At step t, we define N_m^t as the number of operations assigned to M_m since its last maintenance, $O_m^{\text{last},t}$ as the most recently assigned operation on M_m, and AT_m^t as the base available time of M_m, defined as $C_{O_m^{\text{last},t}}$ if at least one operation has been assigned to M_m before step t, and 0 otherwise. The EST and EFT of $e_{ijm} \in \mathcal{A}_t$ in our setting are computed as follows:

$$\text{EST}_{ijm}^t = \max(C_{(i-1)j} + \underbrace{TT_{m_{(i-1)j}m}}_{\delta_{ij}^t},\ AT_m^t + \underbrace{\mathbb{I}\{N_m^t = MI_m\}MD_m + ST_{O_m^{\text{last},t}O_{ijm}}}_{\delta_m^t}), \quad (10)$$

$$\text{EFT}_{ijm}^t = \text{EST}_{ijm}^t + PT_{ijm}. \quad (11)$$

If $i = 1$, $C_{(i-1)j}$ and $TT_{m_{(i-1)j}m}$ are set to RT_j and zero, respectively. For variants in which some constraints are inactive, the corresponding terms are omitted from Equation (10). Note that the subscripts of the delay terms indicate which component of Equation (1) they modify (operation readiness, machine availability, or pair feasibility), not the indices on which they depend.

Our constraint unification readily accommodates diverse operational requirements. For instance, if setup can begin only after both the operation and the machine are ready, the EST computation can be modified as follows:

$$\text{EST}_{ijm}^t = \max(C_{(i-1)j} + \underbrace{TT_{m_{(i-1)j}m}}_{\delta_{ij}^t},\ AT_m^t + \underbrace{\mathbb{I}\{N_m^t = MI_m\}MD_m)}_{\delta_m^t} + \underbrace{ST_{O_m^{\text{last},t}O_{ijm}}}_{\delta_{ijm}^t}. \quad (12)$$

Alternatively, if maintenance and setup can be performed in parallel, the EST computation can be adjusted as follows:

$$\begin{aligned}\text{EST}_{ijm}^t = \max(C_{(i-1)j} + \underbrace{TT_{m_{(i-1)j}m}}_{\delta_{ij}^t},\\ AT_m^t + \underbrace{\max(\mathbb{I}\{N_m^t = MI_m\}MD_m,\ ST_{O_m^{\text{last},t}O_{ijm}})}_{\delta_m^t}).\end{aligned} \quad (13)$$

Our unification can also handle new temporal constraints. For example, many industrial settings require a minimum time lag $d_{(i-1)j}$ between consecutive operations within the same job for process stabilization, cooling, or drying [44]. Assuming this waiting period must elapse before transportation begins, the constraint can be incorporated as follows:

$$\begin{aligned}\text{EST}_{ijm}^t = \max(C_{(i-1)j} + \underbrace{d_{(i-1)j} + TT_{m_{(i-1)j}m}}_{\delta_{ij}^t},\\ AT_m^t + \underbrace{\mathbb{I}\{N_m^t = MI_m\}MD_m + ST_{O_m^{\text{last},t}O_{ijm}}}_{\delta_m^t}).\end{aligned} \quad (14)$$

Note that computing EST requires no information beyond the instance data and the current partial schedule. EST is simply the start time that the environment would assign to the operation if the

pair were selected. Most scheduling simulators must compute this value anyway to execute an action; the USR merely makes it visible to the policy at decision time. That said, EST is a strictly one-step (myopic) quantity, so the downstream effects of each decision must still be learned by the policy. Thus, although EST is key to cross-variant generalization, it alone is not sufficient for good performance, and the policy architecture also plays an important role (as validated in Section 5).

B.2 Node and Edge Features

Table 3 summarizes the three types of feature vectors used to represent s_t. Here, $\mathbb{I}\{\mathrm{M}\} \in \{0, 1\}$ indicates whether the maintenance constraint is active. Following prior work [31], we employ a compact feature space rather than a large set of handcrafted features. In practice, we re-anchor EST^t_{ijm} for all $e_{ijm} \in \mathcal{A}_t$ by subtracting the current simulation time T^t, defined as the earliest time at which any candidate O-M pair becomes available, so that the model focuses on when each action can start relative to the current time. EFT is then computed based on the shifted EST. Finally, all time-related features are normalized by the maximum processing time of the instance, $\max_{i,j,m} PT_{ijm}$, to keep their scales comparable across instances.

Table 3: Operation, machine, and O-M pair features for SCHEDFORMER.

Category	Feature	Description
Operation feature $\boldsymbol{f}^t_{O_{ij}} \in \mathbb{R}^2$	Scheduled	1 if O_{ij} has been scheduled; 0 otherwise
	Rem. Proc. time	Remaining time until completion ($C_{ij} - T^t$) if scheduled; otherwise, $\min_{M_m \in \mathcal{M}_{ij}} PT_{ijm}$
Machine feature $\boldsymbol{f}^t_{M_m} \in \mathbb{R}^4$	Busy	1 if M_m is not available; 0 otherwise
	Time to idle	Remaining time until the machine becomes available
	Rem. Ops. to Maint.	Number of operations until maintenance if $\mathbb{I}\{\mathrm{M}\} = 1$; -1 otherwise
	Maint. Dur.	Fixed maintenance duration if $\mathbb{I}\{\mathrm{M}\} = 1$; 0 otherwise
O-M pair feature $\boldsymbol{f}^t_{e_{ijm}} \in \mathbb{R}^3$	EST	Earliest Start Time EST^t_{ijm}, defined in Equation (10)
	Proc. time	Processing time PT_{ijm}
	EFT	Earliest Finish Time EFT^t_{ijm}, defined in Equation (11)

B.3 Pre-Norm Transformer

The original Transformer architecture [32] applies normalization after the residual addition in each sublayer (i.e., the attention and feed-forward sublayers), a design known as Post-Norm:

$$\text{Post-Norm}(\boldsymbol{x}) = \text{Norm}(\boldsymbol{x} + \text{Sublayer}(\boldsymbol{x})). \tag{15}$$

However, recent models in language modeling [58] and computer vision [59] commonly favor Pre-Norm architectures, where normalization is applied to the input of each sublayer within the residual branch to enhance gradient flow, which often yields better performance [45]:

$$\text{Pre-Norm}(\boldsymbol{x}) = \boldsymbol{x} + \text{Sublayer}(\text{Norm}(\boldsymbol{x})). \tag{16}$$

Pre-Norm has also been increasingly adopted in the NCO field, where its advantage has been empirically validated [40]. In our model, Pre-Norm likewise achieves slightly better performance than Post-Norm, as shown in Table 4. We therefore adopt Pre-Norm as the default design of our Transformer blocks.

Table 4: Pre-Norm vs. Post-Norm. Gaps (%) are averaged over 16 variants for each problem size and reported relative to SF (16V). SF (16V) denotes SCHEDFORMER trained on 10×5 instances across all 16 variants, as defined in Section 5.1. Post-Norm follows the same training setting as SF (16V).

Method	10×5 Gap (↓)	20×5 Gap (↓)	20×10 Gap (↓)
Post-Norm	0.27%	0.22%	0.49%
Pre-Norm (SF (16V))	**0.00%**	**0.00%**	**0.00%**

B.4 Solver Design Analysis

To validate our choice of placing CABs before SABs, we compare against two alternatives: 1) reversing the order (SABs→CABs), and 2) running all four blocks in parallel and fusing their outputs via concatenation and linear projection (Parallel). As shown in Table 5, CABs→SABs achieves the best performance, suggesting that incorporating edge information before the SABs is beneficial.

Table 5: Effect of CAB and SAB ordering on solver performance. Gaps (%) are averaged over 16 variants for each problem size and reported relative to SF (16V). Both ablated models follow the same training setting as SF (16V).

Method	10×5 Gap (↓)	20×5 Gap (↓)	20×10 Gap (↓)
SABs→CABs	0.55%	0.32%	0.41%
Parallel	0.57%	0.52%	0.30%
CABs→SABs (SF (16V))	**0.00%**	**0.00%**	**0.00%**

B.5 Multi-Head Self-Attention

As noted in Section 4.2, the outputs of the two CABs, the operation embeddings $\bar{\boldsymbol{h}}_{O_{ij}}^{(l)}$ and the machine embeddings $\bar{\boldsymbol{h}}_{M_m}^{(l)}$, are fed into SABs to capture relationships between nodes of the same type. Here, we present the standard MHSA procedure step by step for $\mathcal{G}_{M2M}$ as an example. The MHSA begins by projecting the normalized embeddings, $\text{Norm}(\bar{\boldsymbol{h}}_{M_m}^{(l)})$, into queries, keys, and values across H attention heads (for readability, the graph-type and step indices are omitted):

$$\boldsymbol{q}_m^{(l)h} = \boldsymbol{W}_q^{(l)h}\text{Norm}(\bar{\boldsymbol{h}}_{M_m}^{(l)}),\ \boldsymbol{k}_m^{(l)h} = \boldsymbol{W}_k^{(l)h}\text{Norm}(\bar{\boldsymbol{h}}_{M_m}^{(l)}),\ \boldsymbol{v}_m^{(l)h} = \boldsymbol{W}_v^{(l)h}\text{Norm}(\bar{\boldsymbol{h}}_{M_m}^{(l)}), \quad (17)$$

where $\boldsymbol{W}_q^{(l)h}$, $\boldsymbol{W}_k^{(l)h}$, and $\boldsymbol{W}_v^{(l)h} \in \mathbb{R}^{d_h \times d}$ are learnable projection matrices for head h. Next, we compute the attention weights $a_{mm'}^{(l)h}$ for each head via a scaled dot product between the queries and keys, followed by a softmax operation:

$$a_{mm'}^{(l)h} = \text{Softmax}_{M_{m'} \in \mathcal{M}}\left(\frac{\langle \boldsymbol{q}_m^{(l)h}, \boldsymbol{k}_{m'}^{(l)h}\rangle}{\sqrt{d_h}}\right). \quad (18)$$

Machines with no remaining processable operations are masked out. Each attention head then produces an output by computing a weighted sum of the projected values: $\hat{\boldsymbol{h}}_{M_m}^{(l)h} = \sum_{M_{m'} \in \mathcal{M}} a_{mm'}^{(l)h} \boldsymbol{v}_{m'}^{(l)h}$. The hidden embeddings from all attention heads are concatenated and linearly projected using a learned matrix $\boldsymbol{W}_f^{(l)} \in \mathbb{R}^{d \times d}$ to generate the output of the MHSA sublayer:

$$\text{MHSA}(\text{Norm}(\bar{\boldsymbol{h}}_{M_m}^{(l)}), \text{Norm}(\bar{\boldsymbol{h}}_{\mathcal{M}}^{(l)})) = \boldsymbol{W}_f^{(l)}\text{Concat}(\hat{\boldsymbol{h}}_{M_m}^{(l)1}, \ldots, \hat{\boldsymbol{h}}_{M_m}^{(l)H}). \quad (19)$$

For $\mathcal{G}_{O2O}$, we integrate RoPE into standard MHSA by applying it to the query $\boldsymbol{q}_{ij}^{(l)h}$ and key $\boldsymbol{k}_{i'j}^{(l)h}$ vectors for each head h to encode relative operation ordering within each job:

$$a_{ii'j}^{(l)h} = \text{Softmax}_{O_{i'j} \in \mathcal{O}_j}\left(\frac{\langle \boldsymbol{R}_i \boldsymbol{q}_{ij}^{(l)h}, \boldsymbol{R}_{i'} \boldsymbol{k}_{i'j}^{(l)h}\rangle}{\sqrt{d_h}}\right). \quad (20)$$

Here, $\boldsymbol{R}_i$ represents a position-dependent rotary transformation that applies a fixed sinusoidal rotation to query and key vectors. Unlike additive Positional Embeddings (PE), which inject absolute position information only at the input level [32], RoPE embeds relative positional information into every attention computation for $\mathcal{G}_{O2O}$ without additional parameters, enhancing order-reasoning capability while mitigating position dilution across layers [48]. As shown in Table 6, when trained on 10×5 instances ($n_j = 5$), the model with RoPE generalizes well to unseen job route lengths ($n_j = 10$), whereas the PE version exhibits a performance drop.

We further examine the attention behavior. Figure 6 visualizes the $\overrightarrow{\text{MHSA}}$ weights for $\mathcal{G}_{O2O}$. Different heads exhibit distinct attention patterns: some concentrate along and around the diagonal, focusing on nearby operations, while others attend more broadly across the sequence. These patterns remain qualitatively consistent across jobs. Notably, the diagonal attention patterns align with the successor and predecessor relations that existing works inject as an explicit inductive bias [29, 30], while the broader patterns capture complementary views that such handcrafted biases do not encode.

Table 6: Ablation study on RoPE. Gaps (%) are averaged over 16 variants for each problem size and reported relative to SF (16V). Both ablated models follow the same training setting as SF (16V).

Method	10×5 Gap (↓)	20×5 Gap (↓)	20×10 Gap (↓)
w/o_RoPE	0.76%	0.79%	0.95%
PE instead of RoPE	0.60%	0.78%	2.40%
SF (16V)	**0.00%**	**0.00%**	**0.00%**

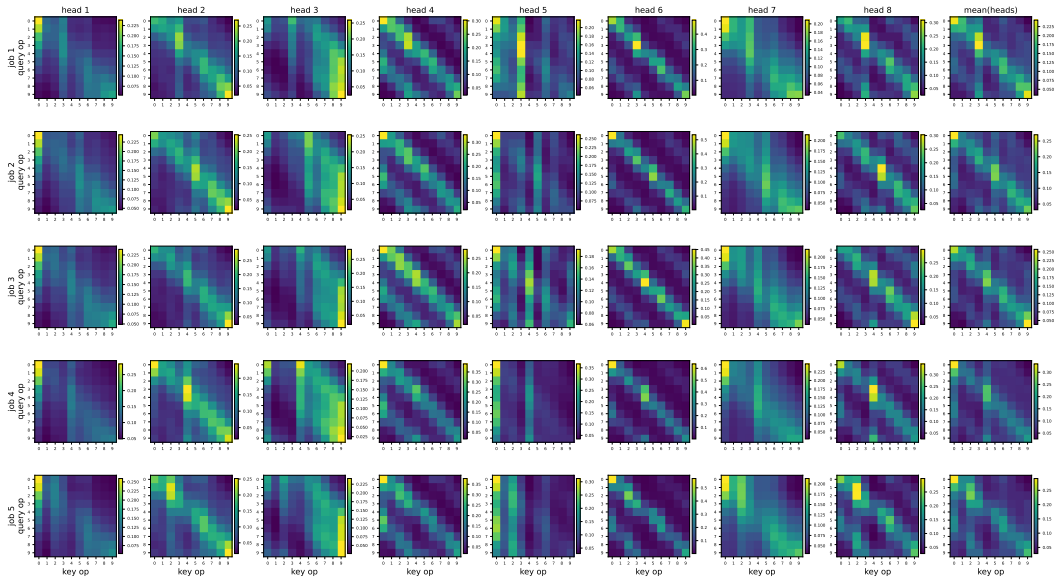


Figure 6: **Attention weights of the SAB for $\mathcal{G}_{O2O}$.** Weights are extracted from the last layer ($l = 2$) at the first decision step ($t = 1$) on the first test instance of the 20×10 classic FJSSP. Rows show the first five jobs, and columns show the eight attention heads and their mean. Each panel maps query operations to key operations within the same job.

B.6 Rationale for Using ASIL

In multi-task learning for VRPs, existing studies predominantly train models with POMO [27], a variant of the REINFORCE algorithm. Concretely, for each instance $\boldsymbol{x}$ in a batch of size B, POMO samples G solutions and uses their average reward as a baseline $b(\boldsymbol{x})$. The policy is then updated using all $B \times G$ trajectories. Retaining the computation graphs for all $B \times G$ trajectories thus incurs substantial memory overhead. This issue is especially severe in neural constructive solvers for scheduling problems, which typically use heavy-decoder architectures that recompute latent embeddings at every decision step to reflect state dynamics. In our training setup and hardware configuration, POMO encounters GPU Out-of-Memory (OOM) errors. Although gradient accumulation can alleviate this issue, it comes at the cost of significantly increased training time.

In contrast, as outlined in Algorithm 1, ASIL requires only the best solution per instance for the model update. Pseudo-labels are first collected via gradient-free forward passes, after which replay-based imitation learning computes gradients only for the best trajectory. Consequently, computation graphs are retained for only B trajectories during the model update, greatly reducing memory consumption and accelerating training. This efficiency enables a larger G under the same hardware and training budget, thereby increasing the likelihood of obtaining higher-quality training labels.

Table 7 empirically shows that increasing G consistently improves performance. Except for G, all training settings are as described in Appendix C.2. Notably, even with $G = 16$, ASIL already outperforms POMO with $G = 128$ while using less than 1/20 of the training time (**3 hours** vs. 67 hours). We hypothesize that ASIL's imitation objective reduces gradient variance relative to trajectory-level RL (POMO), potentially contributing to more stable training. ASIL can also directly

Algorithm 1 Training Algorithm

Input: Model parameters θ, instance distribution $\mathcal{X}$, number of iterations E, batch size B, and rollouts per instance G
for $iteration = 1$ to E **do**
  // Phase 1: Gradient-Free Solution Sampling
  $\boldsymbol{x}_b \leftarrow \texttt{SampleInstance}(\mathcal{X}), \quad \forall b \in \{1, \dots, B\}$
  $\{\boldsymbol{\tau}_b^1, \dots, \boldsymbol{\tau}_b^G\} \leftarrow \texttt{SampleRollout}(\boldsymbol{x}_b, \pi_\theta), \quad \forall b \in \{1, \dots, B\}$
  $\boldsymbol{\tau}_b^* \leftarrow \arg\min_{\boldsymbol{\tau}_b^g \in \{\boldsymbol{\tau}_b^1, \dots, \boldsymbol{\tau}_b^G\}} C_{\max}(\boldsymbol{\tau}_b^g, \boldsymbol{x}_b), \quad \forall b \in \{1, \dots, B\}$ ▷ Select the best solution
  $A(\boldsymbol{x}_b) \leftarrow \texttt{ComputeAdvantage}(\{C_{\max}(\boldsymbol{\tau}_b^g, \boldsymbol{x}_b)\}_{g=1}^G), \quad \forall b \in \{1, \dots, B\}$
  // Phase 2: Replay-Based Imitation Learning
  $\texttt{ReplayBestSolution}(\boldsymbol{x}_b, \pi_\theta, \boldsymbol{\tau}_b^*), \quad \forall b \in \{1, \dots, B\}$ ▷ Replay the best solution
  $\mathcal{L}_{\text{ASIL}} = -\frac{1}{B|\mathcal{O}|} \sum_{b=1}^{B} A(\boldsymbol{x}_b) \sum_{t=1}^{|\mathcal{O}|} \log \pi_\theta(a_{b,t}^* \mid s_{b,t}^*, \boldsymbol{x}_b)$ ▷ Next-action prediction
  $\theta \leftarrow \text{AdamW}(\theta, \nabla_\theta \mathcal{L}_{\text{ASIL}})$ ▷ Update model
end for
Output: Trained parameters θ

Table 7: Effects of G and training algorithms on model performance. Gaps (%) are averaged over 16 variants for each problem size and reported relative to SF (16V) trained with ASIL and $G = 128$.

Method	G	**Algorithm**	10×5 Gap (↓)	20×5 Gap (↓)	20×10 Gap (↓)	Training time (↓)
SF (16V)	128	POMO	1.93%	1.82%	2.11%	67h
SF (16V)	16	ASIL	1.69%	1.71%	1.40%	**3.0h**
SF (16V)	32	ASIL	1.22%	0.61%	0.88%	4.5h
SF (16V)	64	ASIL	0.94%	0.29%	0.34%	8.4h
SF (16V)	128	ASIL	**0.00%**	**0.00%**	**0.00%**	15.2h

distill useful solution patterns into the solver from promising rollouts, even with only a few samples. For the POMO implementation, we use gradient accumulation to avoid OOM errors, and further adopt mixed batch training and reward normalization from [40] to improve performance. Based on these results, we use ASIL with $G = 128$ as the default training configuration in all experiments, including for the neural baselines, as it achieves the best performance within a reasonable training budget.

Another common approach in classic FJSSP studies is to train the policy using PPO with dense rewards [20, 29, 30]. These rewards are typically defined by the change in the lower bound of the makespan after each action. However, this lower bound assumes that operations within the same job are processed without waiting, and therefore overlooks resource contention. As a result, the reward may fail to capture the true scheduling impact of an action, especially in more complex FJSSP variants. More informative reward functions could be designed for individual variants, but doing so would require additional engineering effort and careful reward shaping by scheduling experts. We believe such surrogate reward design involves roughly the same effort as building a variant-specific teacher solver to generate labels for SL. ASIL requires neither.

C Experimental Details

This section details the data generation procedure, training setup, and baseline implementation. All experiments are conducted on a 24-core Intel(R) i9-14900KS CPU and a single NVIDIA GeForce RTX 4090 with 24 GB of memory.

C.1 Data Generation Procedure

Classic Flexible Job Shop Scheduling Problem (classic FJSSP). We first generate classic FJSSP instances, which serve as the backbone for all variants, following the SD_2 procedure [30, 31]. Specifically, for a $|\mathcal{J}| \times |\mathcal{M}|$ FJSSP instance $\boldsymbol{x}$, each J_j has $|\mathcal{M}|$ operations. For each O_{ij}, the

values $|\mathcal{M}_{ij}|$ and PT_{ijm} are randomly sampled from discrete uniform distributions $U(1, |\mathcal{M}|)$ and $U(1, 99)$, respectively.

Release Time (R). Following previous studies [60, 61], half of the jobs, selected uniformly at random, have $RT_j = 0$, while the remaining jobs have RT_j sampled from $U(1, 10 \times |\mathcal{J}|)$.

Sequence-Dependent Setup Time (S). Following well-established procedures [10, 62, 63], $ST_{O_{ij}O_{i'j'}m}$ is sampled from $U(1, 50)$, where the upper bound corresponds to approximately 50% of the maximum PT_{ijm}. The setup time of the first operation on each machine is assumed to be zero.

Machine Maintenance (M). Similar to previous work [9], the maintenance interval MI_m and maintenance duration MD_m are sampled from $U(4, 8)$ and $U(1, 124)$, respectively.

Transportation Time (T). Similar to prior work [11], the transportation time $TT_{mm'}$ is sampled from $U(1, 40)$ for $m \neq m'$, while $TT_{mm} = 0$, as no transportation is needed between consecutive operations on the same machine. The first operation O_{1j} of each job J_j is assumed to start directly on its assigned machine without an initial transportation delay.

C.2 Training Setup

In our experiments, all learning-based methods, including baselines, follow the same training setup: $E = 4000$ iterations with a batch size of $B = 50$, for a total of 200,000 training instances. At each iteration, one variant is sampled uniformly at random from the training variants, and B instances of that variant are generated on the fly. During training, we sample $G = 128$ trajectories per instance to generate pseudo-labels. We use the AdamW optimizer [64] with $(\beta_1, \beta_2) = (0.9, 0.95)$ and a weight decay of 0.1. The learning rate is linearly warmed up to 3×10^{-4} over the first 400 iterations ($0.1E$), and then linearly decayed to zero over the remaining iterations. In preliminary experiments, this schedule consistently showed better performance than constant learning rates (10^{-4} and 3×10^{-4}). For ASIL, we set $\epsilon = 10^{-8}$. We use the model checkpoint from the final iteration for testing.

C.3 Baseline Implementation

OR-Tools. As our exact baseline, we use **CP-SAT** from Google OR-Tools v9.15 [50], a solver widely used for CO problems. We implement a constraint programming model for each of the 16 variants. The difficulty of these models varies considerably across variants. The classic FJSSP and the variants involving only R and/or T are comparatively easy for CP-SAT on small instances because their machine-capacity constraints can be represented compactly using optional interval variables and `AddNoOverlap`. Release times impose simple lower bounds on the start time of the first operation of each job, while transportation times introduce assignment-dependent delays between consecutive operations of the same job. Neither constraint requires explicitly identifying the immediate predecessor of each operation in a machine sequence.

By contrast, the S and M constraints cannot be handled by `AddNoOverlap` alone because they depend on the detailed sequence of operations on each machine. Sequence-dependent setup times are determined by an operation's immediate predecessor, while periodic maintenance depends on the operation's position in the sequence. We implement these constraints using a circuit-based formulation via `AddCircuit`: setup times are activated on selected predecessor arcs, while maintenance positions are represented by operation-rank variables propagated along these arcs. Compared with a basic position-indexed formulation, this reduces the number of reified sequencing constraints from $O(U_m^3)$ to $O(U_m^2)$ per machine, where U_m denotes the number of candidate operation-machine assignments on machine M_m. It also benefits from the stronger native propagation of `AddCircuit`.

Nevertheless, as shown in Table 1 of the main text, this efficiency does not overcome the combinatorial explosion of the underlying search space. For the 10×5 instances, CP-SAT finds good solutions, albeit with far longer search times than constructive methods. As instance size grows, CP-SAT fails to find a feasible solution within the time limit for some S/M test instances. When CP-SAT does find feasible solutions for larger S/M instances, their makespans are 2–4 times those of SCHEDFORMER. Longer time budgets would likely help CP-SAT find better solutions. However, on larger classic FJSSP instances (30×10 and 40×10), SCHEDFORMER still outperforms CP-SAT even when the latter is given 1800s per instance, reducing makespan by about 1.6% at 30×10 and 12.5% at 40×10.

Priority Dispatching Rules. PDRs, which select operations based on predefined rules, are widely used in practice due to their flexibility and real-time decision-making ability. We include three representative PDRs as our baselines.

- **FIFO (First-In-First-Out):** Selects the earliest free candidate operation. Here, a candidate operation refers to an unscheduled operation whose predecessors have all been scheduled.
- **MOR (Most Operations Remaining):** Selects the candidate operation with the most remaining successors.
- **MWKR (Most Work Remaining):** Selects the candidate operation with the largest total remaining average processing time over successor operations.

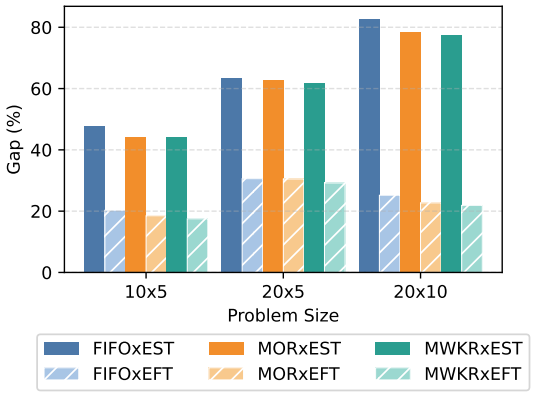


Figure 7: **Performance comparison of EST- and EFT-based machine selection in PDRs.** Gaps (%) are averaged over 16 variants for each problem size and reported relative to SF (16V).

In many PDR implementations [29, 30, 31], the selected operation is assigned to the machine with the lowest EST. However, as shown in Figure 7, using the EFT as the machine selection criterion yields substantially better performance than using the EST, as it better reflects our objective of minimizing the maximum finish time across all operations. Accordingly, we adopt the EFT as the default machine selection criterion in our PDR implementations to establish a strong set of PDR baselines.

Neural Constructive Solvers. We also compare SCHEDFORMER with three representative neural constructive solvers: **HGNN** [29], **DAN** [30], and **ReSched** [31]. However, as mentioned in Section 2.1, these solvers are specifically designed for the classic FJSSP. For a fair comparison, we provide these methods with the same features used in SCHEDFORMER, including EST and EFT as additional edge attributes. Since all baseline solvers already use edge features, no architectural modification is required. We train the baselines using the same multi-task learning setting as SF (16V).

As a sanity check, we compare our retrained baselines with their originally reported results on the widely used 10×5 SD_2 benchmark [30], which contains 100 instances drawn from the same distribution as our classic FJSSP training set. As shown in Table 8, every retrained baseline outperforms its originally reported result.

Table 8: Sanity check results. $\star$: Results are obtained from [31]. (16V): Our retrained version. Gap is the average relative gap to OR-Tools (1800s per instance). Δ: absolute gap difference of (16V) relative to $\star$. **Boldface** indicates the better result in each pair.

Method	10×5 FJSSP	
	Gap (↓)	Δ
HGNN$\star$	71.42%	–
HGNN (16V)	**65.14%**	−6.28
DAN$\star$	25.68%	–
DAN (16V)	**17.21%**	−8.47
ReSched$\star$	16.36%	–
ReSched (16V)	**15.71%**	−0.65

D Additional Experiments

In this section, we present supplementary experimental results and analyses. For all experiments, we directly reuse the trained SF (16V), SF (4V), and SF (1V) from Table 1 in the main paper without any retraining.

D.1 Makespan Results

Table 9 presents the MS results for each method listed in Table 1. The results show that makespan scales differ markedly across problem variants, underscoring the substantial effect of constraint configurations on the resulting makespans.

Table 9: MS results for all compared methods. Notation follows Table 1.

Method	FJSSP			FJSSP-R			FJSSP-S			FJSSP-M		
	10×5	20×5	20×10	10×5	20×5	20×10	10×5	20×5	20×10	10×5	20×5	20×10
OR-Tools	325.15	599.79	482.21	342.65	602.94	506.71	508.14	×	×	431.33	×	×
FIFO	421.95	764.84	639.66	437.48	769.66	670.64	654.72	1259.97	1102.96	521.18	969.66	840.77
MOR	418.38	768.63	625.69	429.83	759.15	653.37	648.72	1276.80	1096.91	510.48	975.98	825.53
MWKR	410.13	747.96	619.17	426.90	745.62	654.68	648.08	1255.91	1086.70	502.15	955.78	824.12
HGNN	532.25	1026.00	1059.12	544.79	1038.51	1059.34	771.04	1505.29	1558.42	605.82	1191.88	1218.87
DAN	383.50	664.59	574.50	398.85	675.71	607.69	567.74	970.06	923.98	470.29	842.56	749.90
ReSched	381.07	655.45	555.74	397.46	666.05	591.50	560.23	916.76	870.56	463.05	838.98	732.95
SF (16V)	356.56	**624.45**	515.47	370.52	**631.03**	553.82	**521.48**	887.41	**814.58**	**428.71**	793.69	675.32
SF (4V)	352.99	625.92	**510.33**	**370.42**	632.14	**552.14**	524.78	**886.96**	825.91	432.61	**792.98**	**671.48**
SF (1V)	**352.09**	628.93	514.31	371.19	633.01	567.46	535.35	895.12	845.70	441.42	817.02	701.45

Method	FJSSP-T			FJSSP-RT			FJSSP-RS			FJSSP-RM		
	10×5	20×5	20×10	10×5	20×5	20×10	10×5	20×5	20×10	10×5	20×5	20×10
OR-Tools	349.87	603.98	566.21	377.54	610.37	612.70	522.61	×	×	446.34	×	×
FIFO	453.83	770.44	709.08	474.55	789.86	779.61	663.78	1272.55	1113.69	529.35	975.30	874.69
MOR	441.69	773.48	700.65	464.81	783.37	754.26	660.70	1262.31	1098.02	516.33	961.11	844.68
MWKR	439.47	761.30	697.72	465.40	768.69	725.83	652.92	1259.81	1099.65	513.33	956.39	841.11
HGNN	554.89	1048.50	1149.54	569.59	1055.06	1149.90	778.05	1530.67	1553.32	613.16	1208.12	1227.51
DAN	409.90	672.92	659.21	437.12	689.89	715.07	585.28	983.92	949.14	483.29	852.82	778.76
ReSched	417.97	691.48	662.62	436.52	692.65	706.68	568.35	937.57	897.01	473.38	847.47	756.17
SF (16V)	389.11	628.39	607.24	**413.32**	**640.03**	677.22	**538.40**	**899.46**	849.83	**445.75**	801.55	**697.51**
SF (4V)	**387.74**	**628.03**	**606.12**	413.89	641.06	**676.80**	539.35	902.16	**844.22**	446.05	**800.19**	705.89
SF (1V)	389.03	629.85	615.42	414.33	640.50	691.13	547.53	911.10	865.94	456.33	821.38	728.15

Method	FJSSP-SM			FJSSP-ST			FJSSP-MT			FJSSP-RSM		
	10×5	20×5	20×10	10×5	20×5	20×10	10×5	20×5	20×10	10×5	20×5	20×10
OR-Tools	636.66	×	×	523.83	×	×	449.57	×	×	655.68	×	×
FIFO	755.28	1474.17	1316.50	667.37	1260.68	1124.53	533.33	971.11	887.38	762.13	1465.51	1332.53
MOR	742.44	1469.67	1303.53	661.93	1278.89	1109.89	525.12	979.28	870.24	754.33	1465.50	1313.45
MWKR	735.19	1466.97	1293.53	663.03	1258.50	1099.05	518.24	965.67	866.48	742.60	1465.45	1307.64
HGNN	846.89	1684.66	1723.61	774.31	1514.12	1564.89	627.98	1214.81	1297.02	861.62	1711.27	1736.49
DAN	660.69	1155.63	1108.47	600.29	982.09	988.82	491.17	849.06	807.83	667.78	1170.89	1136.64
ReSched	644.03	1110.30	1054.67	593.62	949.17	956.44	495.69	857.96	809.66	655.46	1118.60	1079.65
SF (16V)	**601.67**	**1055.81**	**989.54**	**543.86**	**902.86**	**890.94**	456.08	**795.03**	**736.95**	**614.92**	**1073.03**	**1015.96**
SF (4V)	606.60	1063.42	1007.28	551.47	903.19	897.08	**455.34**	797.76	737.57	622.97	1079.35	1029.63
SF (1V)	634.71	1094.66	1038.10	560.58	915.58	917.84	470.05	821.12	763.69	642.63	1109.08	1056.50

Method	FJSSP-RST			FJSSP-RMT			FJSSP-SMT			FJSSP-RSMT		
	10×5	20×5	20×10	10×5	20×5	20×10	10×5	20×5	20×10	10×5	20×5	20×10
OR-Tools	542.54	×	×	463.44	×	×	639.83	×	×	668.49	×	×
FIFO	681.59	1275.57	1151.02	556.67	982.24	949.60	757.91	1474.52	1327.87	771.95	1470.89	1364.26
MOR	670.42	1260.82	1126.11	540.83	989.38	908.34	754.56	1469.25	1319.39	763.15	1469.54	1325.66
MWKR	666.53	1265.08	1121.21	543.97	969.39	902.29	743.78	1472.87	1312.52	754.93	1470.66	1318.66
HGNN	779.26	1547.01	1566.97	642.23	1226.76	1305.12	847.57	1687.91	1736.23	862.94	1710.94	1746.64
DAN	611.97	1004.35	1023.38	515.41	868.39	856.82	683.68	1169.56	1156.55	696.31	1184.13	1193.18
ReSched	604.25	969.16	997.46	512.99	865.32	850.64	672.30	1131.26	1121.58	686.87	1146.83	1158.84
SF (16V)	**567.71**	**922.90**	**921.82**	**477.62**	**802.55**	**790.05**	**624.24**	**1071.67**	**1043.94**	**639.03**	**1089.28**	**1077.59**
SF (4V)	568.60	923.99	930.86	478.51	803.39	802.89	630.88	1077.65	1074.54	655.48	1098.38	1100.21
SF (1V)	580.85	939.33	958.98	487.98	832.26	810.67	655.66	1105.01	1097.22	660.24	1131.83	1121.08

D.2 Generalization under Distribution Shifts

In this section, we study the cross-distribution generalization ability of SCHEDFORMER. Specifically, test instances are generated from distributions where the upper bound of constraint values is scaled by factors of 1.5 (Medium) and 2 (High) relative to the training distributions. For instance, under the High scenario, $ST_{O_{ij}O_{i'j'}m}$ is randomly drawn from $U(1,100)$ rather than $U(1,50)$.

Figures 8 and 9 report the experimental results. SF (16V) achieves statistically significant improvements over both DAN and ReSched across all 15 constrained variants in both scenarios ($p < 0.001$). These results underscore the robustness of SCHEDFORMER under distribution shifts.

D.3 Comparison with a Metaheuristic

In this section, we compare SCHEDFORMER with a metaheuristic algorithm. As a baseline, we select a Biased Random-Key Genetic Algorithm (BRKGA) [65], a strong anytime metaheuristic whose problem-agnostic decoder allows a single implementation to cover all 16 variants, whereas other metaheuristic methods, such as tabu search and large neighborhood search, typically require variant-specific neighborhoods and tuning. Following [66], random keys define a precedence-feasible operation sequence, machines are selected by a greedy EFT rule, and all variants are evaluated using

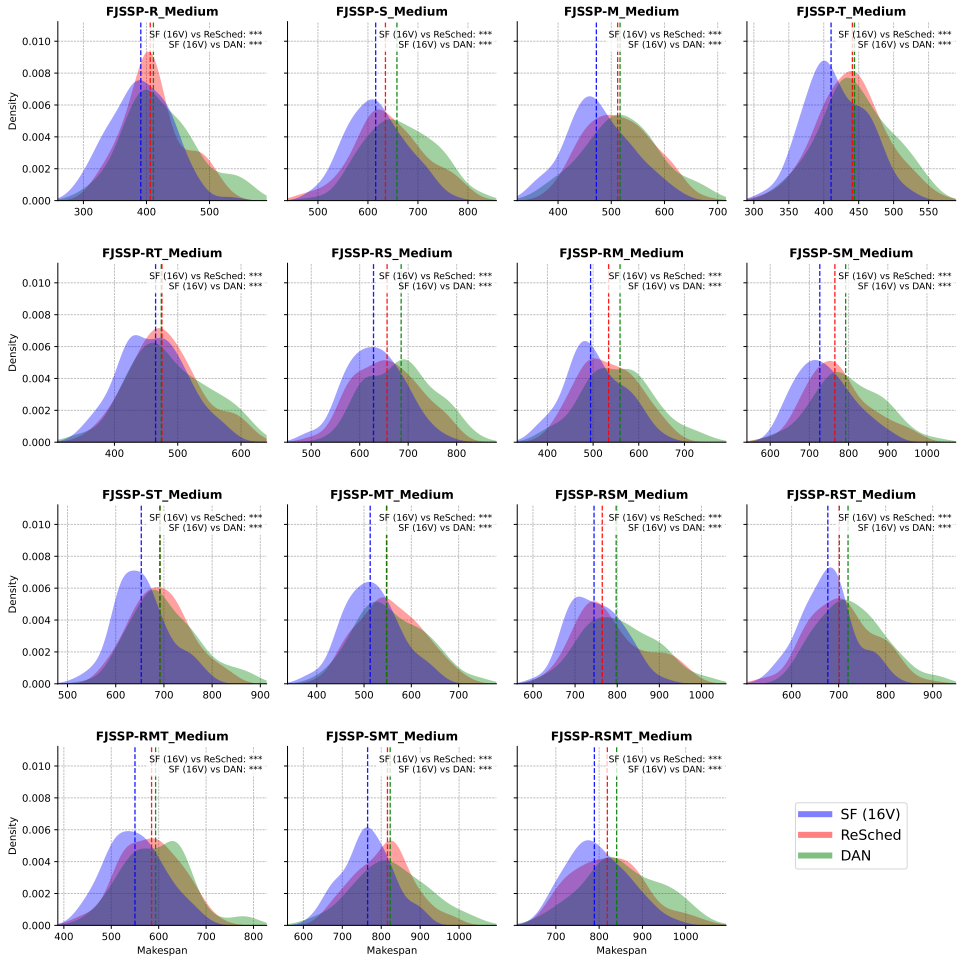


Figure 8: **Performance under the Medium** ($1.5\times$) **distribution shift.** Density plots show the makespan distribution on 10×5 instances, with 100 instances per variant, and dashed lines indicate the median of each method. *** indicates statistical significance at the 0.1% level ($p < 0.001$), based on the two-sided Wilcoxon signed-rank test.

Table 10: Comparison with BRKGA. Gaps (%) are averaged over 16 variants and reported relative to SF (16V).

Method	10×5		20×5		20×10	
	Gap (↓)	Time (↓)	Gap (↓)	Time (↓)	Gap (↓)	Time (↓)
BRKGA	**-1.79%**	60s	14.95%	120s	30.87%	180s
SF (16V)	0.00%	0.33s	**0.00%**	0.63s	**0.00%**	1.28s

our simulator without modifying the decoder. We use the reference hyperparameters from [66], with time limits of 60, 120, and 180s for the 10×5, 20×5, and 20×10 instances, respectively.

Table 10 reports the results. While BRKGA attains better solutions on the smallest 10×5 instances, it requires substantially more computation due to its population-based search. On the larger 20×5 and 20×10 instances, however, SCHEDFORMER outperforms BRKGA on all 16 variants while being over $100\times$ faster. Moreover, with simple sampling-based decoding (see Appendix D.5), SCHEDFORMER surpasses BRKGA even on 10×5 instances within one second, making it particularly attractive when schedules must be produced quickly or repeatedly. These results therefore suggest SCHEDFORMER's potential for real-world scheduling under tight time budgets.

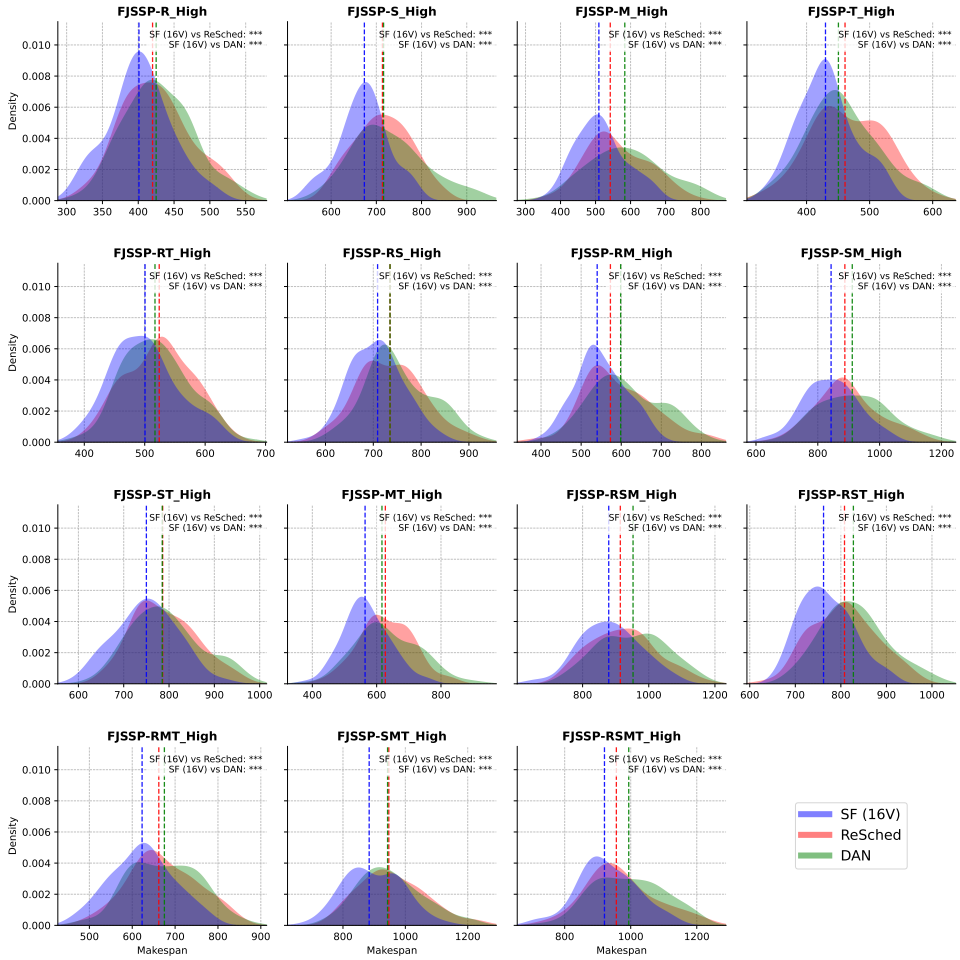


Figure 9: **Performance under the High (2×) distribution shift.** Density plots show the makespan distribution on 10×5 instances, with 100 instances per variant, and dashed lines indicate the median of each method. *** indicates statistical significance at the 0.1% level ($p < 0.001$), based on the two-sided Wilcoxon signed-rank test.

Table 11: Scalability to larger instances. Gaps (%) are averaged over 16 variants and reported relative to SF (16V). For each variant, we evaluate 100 instances of size 40×10 and 20 instances at each of the sizes 50×20 and 100×20.

Method	40×10		50×20		100×20	
	Gap (↓)	Time (↓)	Gap (↓)	Time (↓)	Gap (↓)	Time (↓)
DAN	7.70%	1.78s	6.97%	6.19s	8.41%	16.21s
ReSched	6.32%	1.36s	9.01%	5.32s	8.37%	12.25s
SF (16V)	**0.00%**	2.59s	**0.00%**	8.97s	**0.00%**	19.72s

D.4 Scalability to Larger Instances

We further compare SF (16V) with DAN and ReSched on larger instances of sizes 40×10, 50×20, and 100×20 across all 16 FJSSP variants. As shown in Table 11, SF (16V) achieves the best average solution quality even on instances with 40 times more operations than those seen in training, demonstrating strong cross-size generalization. This quality, however, comes at a higher computational cost because processing the four relation graphs becomes more expensive as instance size grows. A promising remedy, inspired by locality-based attention in large-scale neural routing [67], is to prune nodes from the relation graphs. For example, a lookahead window that retains each

job's currently dispatchable operation and a few subsequent operations could reduce the graph size and attention cost. We leave this for future work.

D.5 Sampling Performance

As a stochastic constructive policy, SCHEDFORMER readily benefits from sampling-based decoding, a simple form of test-time search. We evaluate this strategy on all datasets from Table 1 by sampling 100 solutions per instance and reporting the best one. For comparison, we apply the same strategy to ReSched, the strongest neural baseline.

As shown in Table 12, sampling improves both solvers, yet SF (16V)-S consistently maintains its advantage over ReSched-S. Notably, on the unseen 20×5 and 20×10 sizes, SF (16V)-G still outperforms ReSched-S despite using a single greedy rollout against 100 samples, highlighting the strength of the learned policy itself rather than sampling alone.

Table 12: Sampling results. -G: Greedy strategy. -S: Sampling strategy. Gaps (%) are averaged over 16 variants and reported relative to SF (16V)-G. **Boldface** indicates the best result for each decoding strategy.

Method	10×5		20×5		20×10	
	Gap (↓)	Time (↓)	Gap (↓)	Time (↓)	Gap (↓)	Time (↓)
ReSched-G	7.18%	0.27s	5.87%	0.54s	7.40%	1.06s
ReSched-S	-1.70%	0.36s	3.20%	0.95s	4.76%	4.22s
SF (16V)-G	**0.00%**	0.33s	**0.00%**	0.63s	**0.00%**	1.28s
SF (16V)-S	**-7.04%**	0.41s	**-3.98%**	1.10s	**-4.77%**	4.83s

D.6 The Hurink Benchmark

The Hurink dataset [52] is divided into three categories, edata, rdata, and vdata, each consisting of 40 instances. Each category is further grouped into eight problem sizes ranging from 10×5 to 30×10, with five instances per size. Table 13 shows that these three sets differ in two key statistics: the average number of assignable machines per operation (avg$|\mathcal{M}_{ij}|$) and the maximum number of assignable machines for an operation (max$|\mathcal{M}_{ij}|$).

Table 13: Statistics of the Hurink dataset.

edata		rdata		vdata	
avg$\lvert\mathcal{M}_{ij}\rvert$	max$\lvert\mathcal{M}_{ij}\rvert$	avg$\lvert\mathcal{M}_{ij}\rvert$	max$\lvert\mathcal{M}_{ij}\rvert$	avg$\lvert\mathcal{M}_{ij}\rvert$	max$\lvert\mathcal{M}_{ij}\rvert$
1.15	2 (if $\lvert\mathcal{M}\rvert < 10$) 3 (if $\lvert\mathcal{M}\rvert \geq 10$)	2	3	$\frac{1}{2}\lvert\mathcal{M}\rvert$	$\frac{4}{5}\lvert\mathcal{M}\rvert$

For every instance, the number of operations per job n_j is equal to the number of machines $|\mathcal{M}|$, and each operation has identical processing times across its assignable machines, i.e., $PT_{ijm} = PT_{ijm'}, \forall M_m, M_{m'} \in \mathcal{M}_{ij}$. PT_{ijm} is sampled from $U(5, 99)$. Therefore, compared with our training set, the Hurink dataset has 1) larger, unseen instances, 2) a different processing time distribution, 3) low machine flexibility in edata and rdata, and 4) identical processing times across all eligible machines of each operation.

For the baselines, HGNN [29], DAN [30], DOAGNN [20], ReSched [31], ME-GNN [23], DDGARL [24], and MCGA [22] are trained on 10×5 instances following the SD_1 generation procedure [29]. Models trained on SD_1 have been reported to perform better on the Hurink benchmarks than those trained on SD_2 [30]. LMDRL [21] is trained on 10×5 instances as well, but with processing times sampled from $U(1, 99)$. MPGN [18] trains a separate model for each problem size: 10×5, 10×10, and 15×15. Each model is applied to test instances with the corresponding number of machines.

Experimental results are reported in Table 14. Across all three Hurink benchmarks, every SCHEDFORMER version substantially outperforms all specialized neural baselines. Notably, SF

Table 14: Experimental results on the Hurink benchmark. Gap: Average relative gap to the best-known solution [53]. $\star$: Results are obtained from the original papers. All methods use greedy decoding at test time. **Boldface** and underlining indicate the best and second-best results among neural solvers.

Method	edata Gap (↓)	rdata Gap (↓)	vdata Gap (↓)
HGNN$\star$	15.53%	11.15%	4.25%
MPGN$\star$	23.01%	16.51%	6.96%
DAN$\star$	16.33%	11.42%	3.28%
DOAGNN$\star$	16.21%	11.83%	4.32%
LMDRL$\star$	15.54%	12.09%	5.37%
ME-GNN$\star$	16.58%	14.82%	5.59%
MCGA$\star$	16.06%	10.86%	4.18%
ReSched$\star$	15.48%	10.18%	3.48%
DDGARL$\star$	16.25%	11.15%	3.62%
SF (16V)	10.29%	8.02%	2.21%
SF (4V)	**9.42%**	7.78%	2.32%
SF (1V)	9.87%	**7.74%**	**2.11%**

(16V) and SF (4V) also perform comparably to SF (1V), which is trained only on the classic FJSSP, showing little loss from multi-task training.

D.7 The Behnke Benchmark

In this section, we evaluate SCHEDFORMER on the large-scale Behnke FJSSP benchmark [53]. This benchmark reflects realistic industrial settings by grouping similar machines into work centers. Here, similarity means that if an operation is assignable to one machine within a work center, it is also assignable to all machines in that center. We summarize the main characteristics of the benchmark below:

- For each operation, the number of feasible work center assignments is randomly determined. Some operations can be processed by multiple work centers, whereas others are restricted to a single center.
- Unlike the Hurink dataset, which assumes machine-independent processing times, the Behnke dataset models machine-dependent processing times, similar to the SD_2 instance generation procedure. PT_{ijm} is randomly sampled from $U(10, 30)$.
- The number of jobs $|\mathcal{J}|$ is set to 20, 50, or 100, while the number of work centers is set to 5, 10, or 15. Each work center consists of four machines, resulting in total machine counts $|\mathcal{M}|$ of 20, 40, and 60. Therefore, the problem sizes range from 20×20 to 100×60. Each problem size class consists of five instances.
- In all instances, each job has five operations.

Consequently, compared with our training set, the Behnke dataset has 1) significantly larger, unseen instances and 2) a different processing time distribution.

For evaluation, we compare SCHEDFORMER with representative neural constructive solvers, including HGNN [29], DAN [30], MPGN [18], and EDSP [19]. All baselines are single-task models designed for the classic FJSSP, and their training setups differ. EDSP is trained on instances ranging from 5×4 to 15×13 with processing times sampled from $U(8, 25)$. MPGN trains a separate model for each problem size: 20×20, 20×40, and 20×60. Each model is applied to test instances with the corresponding number of machines. The HGNN and DAN results are taken directly from [19], whose authors obtained them with the official open-source implementations but did not report the evaluation settings. As a reference, we use the CPLEX CP results (3600s limit per instance) from [53] and report the relative gap in makespan to this reference (Gap).

Table 15: Experimental results on the Behnke dataset. ⋆: Results are obtained from [18] and [19]. Gap denotes the percentage gap relative to the CPLEX solution. **Boldface** indicates the best results among neural solvers.

Method	**CPLEX**	**HGNN⋆**		**DAN⋆**		**MPGN⋆**		**EDSP⋆**		**SF (16V)**		**SF (4V)**		**SF (1V)**	
	MS (↓)	MS (↓)	Gap (↓)	MS (↓)	Gap (↓)	MS (↓)	Gap (↓)	MS (↓)	Gap (↓)	MS (↓)	Gap (↓)	MS (↓)	Gap (↓)	MS (↓)	Gap (↓)
20 × 20															
1	131	195	48.9%	166	26.7%	143	9.2%	141	**7.6%**	144	9.9%	141	**7.6%**	142	8.4%
2	130	199	53.1%	162	24.6%	142	9.2%	151	16.2%	138	**6.2%**	142	9.2%	148	13.8%
3	128	198	54.7%	160	25.0%	139	8.6%	138	**7.8%**	142	10.9%	140	9.4%	147	14.8%
4	129	203	57.4%	166	28.7%	144	11.6%	148	14.7%	137	**6.2%**	141	9.3%	139	7.8%
5	133	224	68.4%	164	23.3%	146	9.8%	155	16.5%	141	**6.0%**	144	8.3%	143	7.5%
50 × 20															
6	259	346	33.6%	291	12.4%	250	-3.5%	257	-0.8%	246	**-5.0%**	248	-4.2%	251	-3.1%
7	251	367	46.2%	276	10.0%	247	-1.6%	250	-0.4%	241	**-4.0%**	243	-3.2%	242	-3.6%
8	252	367	45.6%	272	7.9%	249	-1.2%	273	8.3%	245	**-2.8%**	246	-2.4%	248	-1.6%
9	258	386	49.6%	282	9.3%	257	-0.4%	258	0.0%	259	0.4%	249	**-3.5%**	260	0.8%
10	262	376	43.5%	282	7.6%	255	-2.7%	269	2.7%	253	-3.4%	254	-3.1%	243	**-7.3%**
100 × 20															
11	566	637	12.5%	464	-18.0%	437	-22.8%	449	-20.7%	447	-21.0%	434	**-23.3%**	466	-17.7%
12	535	606	13.3%	451	-15.7%	430	-19.6%	447	-16.4%	427	-20.2%	421	**-21.3%**	429	-19.8%
13	555	642	15.7%	476	-14.2%	428	-22.9%	444	-20.0%	420	**-24.3%**	420	**-24.3%**	426	-23.2%
14	532	637	19.7%	473	-11.1%	423	**-20.5%**	447	-16.0%	425	-20.1%	423	**-20.5%**	423	**-20.5%**
15	522	615	17.8%	459	-12.1%	427	-18.2%	442	-15.3%	423	-19.0%	416	**-20.3%**	425	-18.6%
20 × 40															
16	122	187	53.3%	135	10.7%	129	5.7%	128	4.9%	121	**-0.8%**	123	0.8%	125	2.5%
17	132	194	47.0%	145	9.8%	140	6.1%	141	6.8%	135	**2.3%**	135	**2.3%**	140	6.1%
18	123	181	47.2%	135	9.8%	133	8.1%	125	1.6%	132	7.3%	124	**0.8%**	129	4.9%
19	125	185	48.0%	140	12.0%	138	10.4%	134	7.2%	129	3.2%	128	**2.4%**	130	4.0%
20	127	191	50.4%	140	10.2%	142	11.8%	142	11.8%	132	3.9%	130	**2.4%**	138	8.7%
50 × 40															
21	272	353	29.8%	261	-4.0%	271	-0.4%	250	-8.1%	237	**-12.9%**	237	**-12.9%**	239	-12.1%
22	259	346	33.6%	267	3.1%	267	3.1%	241	-6.9%	240	-7.3%	240	-7.3%	234	**-9.7%**
23	245	340	38.8%	261	6.5%	261	6.5%	248	1.2%	237	**-3.3%**	243	-0.8%	268	9.4%
24	265	356	34.3%	268	1.1%	250	-5.7%	245	-7.5%	235	**-11.3%**	235	**-11.3%**	235	**-11.3%**
25	253	355	40.3%	242	-4.3%	249	-1.6%	235	-7.1%	220	-13.0%	219	**-13.4%**	227	-10.3%
100 × 40															
26	531	633	19.2%	422	-20.5%	442	-16.8%	421	-20.7%	398	-25.0%	397	**-25.2%**	404	-23.9%
27	536	599	11.8%	420	-21.6%	444	-17.2%	431	-19.6%	410	-23.5%	406	**-24.3%**	450	-16.0%
28	527	593	12.5%	424	-19.5%	454	-13.9%	429	-18.6%	402	**-23.7%**	402	**-23.7%**	506	-4.0%
29	516	598	15.9%	429	-16.9%	471	-8.7%	423	-18.0%	401	-22.3%	399	**-22.7%**	411	-20.3%
30	521	619	18.8%	443	-15.0%	460	-11.7%	459	-11.9%	439	-15.7%	437	**-16.1%**	510	-2.1%
20 × 60															
31	124	194	56.5%	142	14.5%	145	16.9%	131	5.6%	128	3.2%	121	**-2.4%**	122	-1.6%
32	126	187	48.4%	145	15.1%	144	14.3%	127	**0.8%**	128	1.6%	128	1.6%	133	5.6%
33	134	195	45.5%	147	9.7%	141	5.2%	146	9.0%	138	3.0%	134	**0.0%**	140	4.5%
34	121	194	60.3%	150	24.0%	134	10.7%	129	6.6%	121	**0.0%**	121	**0.0%**	124	2.5%
35	131	180	37.4%	160	22.1%	147	12.2%	145	10.7%	137	4.6%	132	0.8%	131	**0.0%**
50 × 60															
36	259	342	32.0%	244	-5.8%	253	-2.3%	252	-2.7%	230	-11.2%	227	**-12.4%**	230	-11.2%
37	255	354	38.8%	237	-7.1%	242	-5.1%	234	-8.2%	227	**-11.0%**	227	**-11.0%**	229	-10.2%
38	257	358	39.3%	239	-7.0%	256	-0.4%	235	-8.6%	225	-12.5%	223	**-13.2%**	226	-12.1%
39	267	360	34.8%	256	-4.1%	268	0.4%	251	-6.0%	242	-9.4%	237	**-11.2%**	253	-5.2%
40	256	355	38.7%	255	-0.4%	262	2.3%	242	-5.5%	230	**-10.2%**	230	**-10.2%**	231	-9.8%
100 × 60															
41	538	611	13.6%	424	-21.2%	439	-18.4%	419	-22.1%	404	-24.9%	395	**-26.6%**	401	-25.5%
42	535	592	10.7%	430	-19.6%	442	-17.4%	414	-22.6%	403	-24.7%	398	**-25.6%**	468	-12.5%
43	531	624	17.5%	443	-16.6%	442	-16.8%	424	-20.2%	407	-23.4%	402	**-24.3%**	435	-18.1%
44	532	597	12.2%	422	-20.7%	443	-16.7%	426	-19.9%	407	-23.5%	405	**-23.9%**	457	-14.1%
45	537	579	7.8%	438	-18.4%	458	-14.7%	419	-22.0%	410	-23.6%	406	**-24.4%**	423	-21.2%
Mean	306.7	387.8	35.0%	284.6	0.7%	279.6	-2.6%	273.7	-4.6%	262.1	-8.5%	**260.1**	**-9.2%**	272.9	-5.9%

Table 15 reports results on the Behnke benchmark. Although its instance sizes and processing time distributions were not seen during training, SF (4V) matches or improves upon the best existing neural baseline on 43 of 45 instances, while SF (16V) does so on 38 instances.

D.8 Cross-Problem Generalization to the JSSP

The JSSP is a special case of the classic FJSSP in which each operation is assigned to a fixed machine ($|\mathcal{M}_{ij}| = 1$). Consequently, the processing time PT_{ijm} simplifies to PT_{ij}. To evaluate cross-problem generalization to the JSSP, we test SCHEDFORMER on three widely studied JSSP benchmarks: the TA [68], DMU [69], and LA [70] datasets. The TA dataset comprises eight sizes ranging from 15×15 to 100×20, with 10 instances per size. PT_{ij} is sampled from $U(1, 99)$. The DMU dataset consists of eight sizes ranging from 20×15 to 50×20, with 10 instances per size. PT_{ij} is sampled from $U(1, 200)$. The LA dataset comprises eight sizes ranging from 10×5 to 30×10, with five instances per size. PT_{ij} is sampled from $U(5, 99)$. On these benchmarks, we directly use SF (16V), SF (4V), and SF (1V) from Table 1 without any retraining.

We compare SCHEDFORMER with OR-Tools (3600s time limit per instance) and representative neural constructive solvers, including L2D [5], SchN [72], RASCL [73], SLIM [71], DAN [30], and ReSched [31]. The first four baselines are designed and trained for the JSSP, while the last two were originally developed for the classic FJSSP but were retrained from scratch on JSSP instances. Specifically, L2D

Table 16: Experimental results on three JSSP benchmarks. Gap denotes the average relative gap to the best-known solution from [71]. ⋆: Results are obtained from [71] and [31]. Some baselines are omitted for certain benchmarks because no public results are available. **Boldface** and underlining indicate the best and second-best results among neural solvers.

Bench	Method	15×15 Gap (↓)	20×15 Gap (↓)	20×20 Gap (↓)	30×15 Gap (↓)	30×20 Gap (↓)	50×15 Gap (↓)	50×20 Gap (↓)	100×20 Gap (↓)	Mean Gap (↓)
TA	OR-Tools⋆	0.1%	0.2%	0.7%	2.1%	2.8%	3.0%	2.8%	3.9%	2.0%
	L2D⋆	26.0%	30.0%	31.6%	33.0%	33.6%	22.4%	26.5%	13.6%	27.1%
	ReSched⋆	15.7%	19.7%	16.3%	21.5%	22.5%	16.1%	15.6%	9.6%	17.1%
	SchN⋆	15.3%	19.4%	17.2%	19.1%	23.7%	13.9%	13.5%	6.7%	16.1%
	DAN⋆	16.0%	19.7%	18.5%	19.2%	20.9%	13.1%	14.1%	6.4%	16.0%
	RASCL⋆	14.3%	16.5%	17.3%	18.5%	21.5%	12.2%	13.2%	5.9%	14.9%
	SLIM⋆	**13.8%**	**15.0%**	**15.2%**	17.1%	**18.5%**	10.1%	11.6%	5.8%	13.4%
	SF (16V)	17.5%	16.6%	16.1%	16.2%	19.8%	7.6%	11.4%	**0.7%**	13.2%
	SF (4V)	14.4%	15.9%	17.4%	**14.5%**	20.6%	**7.5%**	**10.9%**	1.7%	**12.9%**
	SF (1V)	18.9%	22.2%	18.5%	20.4%	23.6%	12.6%	14.1%	4.7%	16.9%

DMU	Method	20×15 Gap (↓)	20×20 Gap (↓)	30×15 Gap (↓)	30×20 Gap (↓)	40×15 Gap (↓)	40×20 Gap (↓)	50×15 Gap (↓)	50×20 Gap (↓)	Mean Gap (↓)
	OR-Tools⋆	1.8%	1.9%	2.5%	4.4%	4.1%	4.6%	3.8%	4.8%	3.5%
	L2D⋆	39.0%	37.7%	42.0%	39.7%	35.6%	39.6%	36.5%	39.5%	38.7%
	ReSched⋆	23.7%	22.2%	27.8%	28.3%	26.5%	29.2%	26.3%	31.8%	27.0%
	DAN⋆	24.1%	21.1%	27.8%	30.1%	25.2%	28.3%	25.2%	28.6%	26.3%
	SLIM⋆	18.0%	19.4%	21.8%	25.7%	17.5%	22.2%	15.7%	22.4%	20.3%
	SF (16V)	**17.1%**	17.8%	**17.5%**	**22.9%**	**12.3%**	**18.2%**	**10.5%**	**16.4%**	**16.6%**
	SF (4V)	19.2%	**16.8%**	20.3%	23.8%	13.5%	19.7%	12.0%	18.2%	17.9%
	SF (1V)	19.4%	23.0%	23.5%	26.3%	16.8%	23.0%	14.8%	22.3%	21.1%

LA	Method	10×5 Gap (↓)	10×10 Gap (↓)	15×5 Gap (↓)	15×10 Gap (↓)	15×15 Gap (↓)	20×5 Gap (↓)	20×10 Gap (↓)	30×10 Gap (↓)	Mean Gap (↓)
	OR-Tools⋆	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	L2D⋆	14.3%	23.7%	7.8%	27.2%	27.1%	6.3%	24.6%	8.4%	17.4%
	SchN⋆	12.1%	11.9%	2.7%	14.6%	16.1%	3.6%	15.7%	3.1%	10.0%
	SLIM⋆	9.3%	**8.9%**	2.3%	11.6%	13.6%	2.1%	12.1%	2.0%	7.8%
	SF (16V)	7.6%	12.6%	1.1%	**9.9%**	**12.8%**	**0.0%**	12.5%	**0.4%**	**7.1%**
	SF (4V)	**7.4%**	13.9%	**0.2%**	10.2%	15.3%	0.9%	**8.1%**	1.6%	7.2%
	SF (1V)	9.1%	11.7%	1.7%	13.2%	19.2%	0.2%	18.3%	3.9%	9.6%

uses models trained separately for each problem size up to 30×20, and applies the model trained on 30×20 to solve larger instances. SchN generates training instances by sampling $|\mathcal{J}|$ and $|\mathcal{M}|$ from $U(7, 14)$ and $U(2, 5)$, respectively. RASCL trains on instances of five sizes (15×15, 20×15, 20×20, 30×15, and 30×20), while SLIM uses six sizes ($10 \times 10, 15 \times 10, 15 \times 15, 20 \times 10, 20 \times 15$, and 20×20). DAN and ReSched are trained on 30×20 and 10×10 JSSP instances, respectively. All baselines are trained on instances with PT_{ij} sampled from $U(1, 99)$.

Table 16 reports the evaluation results on the three benchmarks. All neural methods use greedy decoding at test time. SCHEDFORMER achieves the best average performance (**Mean** in the table) among neural solvers, showing that the learned policy transfers effectively beyond the FJSSP variants. Specifically, on TA, SLIM performs better on small- to medium-sized instances, whereas SCHEDFORMER takes the lead on 30×15 and on all tested problem sizes with 50 or more jobs. On DMU and LA, whose processing time distributions differ from those used to train both models, SF (16V) outperforms SLIM in 14 of 16 instance groups, indicating greater robustness to distribution shift. SF (16V) also surpasses OR-Tools on TA 100×20 within 10 seconds and finds optimal solutions on LA 20×5. We further observe that SF (16V) and SF (4V), trained across multiple variants, outperform the single-variant SF (1V), suggesting that multi-task training can induce beneficial cross-problem transfer.

D.9 Cross-Problem Generalization to the FFSP

The FFSP is a special case of the classic FJSSP in which all jobs pass through the same sequence of stages and each operation can be processed by any machine in its stage. To evaluate cross-

Table 17: Experimental results on three FFSP datasets, each containing 1000 test instances. Problem sizes are specified as the number of jobs $\times$ the number of stages $\times$ the number of machines per stage. Following the baseline setting, we sample 24 solutions per instance and report the best. $\star$: Results are obtained from the original papers. **Boldface** and underlining indicate the best and second-best results among neural solvers.

Method	FFSP 20×3×4 MS (↓)	FFSP 50×3×4 MS (↓)	FFSP 100×3×4 MS (↓)
MatNet$\star$	27.26	51.52	91.58
PolyNet$\star$	26.71	51.01	91.22
PARCO$\star$	26.47	51.25	91.55
SF (16V)	25.30	49.71	89.97
SF (4V)	25.21	49.42	**89.53**
SF (1V)	**25.07**	**49.38**	90.04

problem generalization to the FFSP, we test SCHEDFORMER on the synthetic datasets of [54], where PT_{ijm} is sampled from $U(2, 9)$. We compare SCHEDFORMER with three SOTA FFSP models: MatNet [54], PolyNet [74], and PARCO [75], each trained separately for each test size. In contrast, SCHEDFORMER is applied zero-shot, without any training on the FFSP.

From Table 17, we can see that SCHEDFORMER achieves the best performance across all problem sizes. Compared with the strongest baseline at each size, the best-performing SCHEDFORMER version reduces the average makespan from 26.47 to 25.07 on $20 \times 3 \times 4$, from 51.01 to 49.38 on $50 \times 3 \times 4$, and from 91.22 to 89.53 on $100 \times 3 \times 4$. Together with the JSSP results, this suggests that SCHEDFORMER learns transferable scheduling policies that remain competitive across problem families.

D.10 Node and Edge Features of Models without Constraint Unification

For w/o_CU, constraints are represented by separate constraint-specific features, as summarized in Table 18. All other settings, including training hyperparameters, are the same as those used for SCHEDFORMER.

Table 18: Operation, machine, and O-M pair features for the w/o_CU models.

Category	**Feature**	**Description**
Operation feature $\boldsymbol{f}^t_{O_{ij}} \in \mathbb{R}^3$	Scheduled	1 if O_{ij} has been scheduled; 0 otherwise
	Rem. Proc. time	Remaining time until completion ($C_{ij} - T^t$) if scheduled; otherwise, $\min_{M_m \in \mathcal{M}_{ij}} PT_{ijm}$
	Release slack	$\max(RT_j - T^t, 0)$
Machine feature $\boldsymbol{f}^t_{M_m} \in \mathbb{R}^4$	Busy	1 if M_m is not available; 0 otherwise
	Time to idle	Remaining time until the machine becomes available
	Rem. Ops. to Maint.	Number of operations until maintenance if $\mathbb{I}\{\mathrm{M}\} = 1$; -1 otherwise
	Maint. Dur.	Fixed maintenance duration if $\mathbb{I}\{\mathrm{M}\} = 1$; 0 otherwise
O-M pair feature $\boldsymbol{f}^t_{e_{ijm}} \in \mathbb{R}^4$	Proc. time	PT_{ijm}
	Setup time	$ST_{O_m^{\mathrm{last},t} O_{ij} m}$
	Trans. time	$TT_{m_{(i-1)j} m}$
	Base EST	$\max(C_{(i-1)j},\ AT^t_m)$ if $e_{ijm} \in \mathcal{A}_t$; -1 otherwise

E Broader Impact

By enabling a single model to serve many FJSSP variants, SCHEDFORMER may lower the barrier to adopting learning-based scheduling, particularly for organizations that lack the expertise to design and maintain variant-specific solutions. However, because SCHEDFORMER is a black-box scheduler, its decisions are difficult to interpret or audit, which may undermine trust in high-stakes settings. Improving the interpretability of learned policies is thus an important direction for future work.