End-to-End Hyper-Relational Information Extraction for Engineering Diagrams via Dynamically Tokenized Relation Transformer

Supplementary Material

A. Detailed Task Implementation of DTRT

In this chapter, we elaborates on the detailed implementation of the proposed Dynamic Tokenized Relation Transformer (DTRT) in three core tasks: Entity Detection, Scene Graph Classification (SGCLS), and Scene Graph Detection (SGDET), together with task-specific optimizations (e.g., dynamic token pruning, contrastive denoising alignment) to enhance efficiency and accuracy for engineering diagram understanding. The source code of our work is now published at https://github.com/Tianyou-Bai/DTRT-End-to-End-Engineering-Diagram-Parsing-in-Scene-Graph.

A.1. Entity Detection

Entity Detection in DTRT aims to localize and classify all entities (e.g., symbols, texts) in engineering diagrams (e.g., P&IDs, EDs) in an anchor-free manner. The core improvements include cross-scale feature fusion and dynamic query refinement, addressing the challenges of small entity missing and ambiguous boundary localization.

A.1.1. Feature Encoding with Dynamic Token Pruning

The input is a raw diagram image $d \in \mathbb{R}^{H\times W\times 3}$ (resized to 2K images or sliced into 800×800 by default). Then, we extend the Swin-T backbone with a Dynamic Token Pruning (DTP) module to reduce redundant computations while retaining critical entity features:

1. Swin-T backbone outputs multi-scale features $F = \{F_1, F_2, F_3, F_4\}$, where $F_l \in \mathbb{R}^{H_l\times W_l\times C}$.
2. Insert MLP scorers at each stage's end: compute local feature $z^l = \text{MLP}(x_{s+1})$ and global feature $z^g = \sum \hat{D}[i]z^l[i] / \sum \hat{D}[i]$, predict retention probability $\pi = \text{Softmax}(\text{MLP}([z^l, z^g]^T))$.
3. Feed x^*_{s+1} and masked tokens to a 3-layer Transformer reconstructor, generate $d_{rc} = \text{Reshape}(W_{inv} \cdot y_{L_r} + b_{inv})$, compute MSE loss $\mathcal{L}_{rc} = \frac{1}{NC}\sum(d_{rc}[i][c] - d[i][c])^2$ to refine scorers.

A.1.2. Entity Prediction and Loss

Pruned feature x^* (stage 4) is fed to a 6-layer entity decoder ($L_e = 6$) with Mixed Query Selection (MQS):

1. Query initialization: $E = \text{LN}(\text{Shuffle}([W_\eta \cdot z_e[\mathcal{I}_e], \Theta_e]))$ ($z_e[\mathcal{I}_e]$ = high-confidence candidates, Θ_e = learnable queries).
2. Activate an entity branch and project refined entity queries E_{L_e} to bounding boxes $\hat{b}_e = \sigma(\text{MLP}_{\text{bbox}}(E_{L_e}))$ and categories $\hat{c}_e = \text{Softmax}(\text{MLP}_{\text{cls}}(E_{L_e}))$ (16 classes for P&IDs, 12 for EDs).
3. Generate noisy samples based on ground-truth entities: positive samples $\mathcal{Y}^+$: Add small Gaussian noise to ground-truth boxes ($\delta_{\text{small}} \sim \mathcal{N}(0, \sigma^2_{\text{small}} \cdot \mathbb{I}_4)$, $\sigma_{\text{small}} = 0.03$), category remains unchanged (c_i); Negative samples $\mathcal{Y}^-$: Add large Gaussian noise to ground-truth boxes ($\delta_{\text{large}} \sim \mathcal{N}(0, \sigma^2_{\text{large}} \cdot \mathbb{I}_4)$, $\sigma_{\text{large}} = 0.15$), category is randomly replaced ($\tilde{c}_i \neq c_i$). Then, calculate cross-entropy loss between refined query predictions and noisy samples, forcing the model to prioritize matching positive samples:$\mathcal{L}_{\text{dn}} = \mathcal{L}_{\text{CE}}(\hat{c}_e, \mathcal{Y}^+) + \mathcal{L}_{\text{CE}}(\hat{c}_e, \mathcal{Y}^-)$where $\hat{c}_e$ is the category prediction of refined entity queries E_{L_e}
4. Total loss: $\mathcal{L}_{\text{ent}} = \lambda_e(\mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{L1}} + \mathcal{L}_{\text{GIoU}}) + \lambda_{\text{rc}}\mathcal{L}_{\text{rc}} + \lambda_{\text{dn}}\mathcal{L}_{\text{dn}}$ ($\lambda_e = 1.0, \lambda_{\text{rc}} = 0.5, \lambda_{\text{dn}} = 0.3$) ($\mathcal{Y}_{\text{cls}}/\mathcal{Y}_{\text{bbox}}$=ground-truth categories/boxes).

We adopt an MLP branch to project entity queries into bounding boxes, this branch is optional and not activated in some tasks (such as SGCLS).

A.2. Scene Graph Classification

SGCLS takes raw images + pre-defined entity boxes and predicts entity categories/hyper-relations (symbol-symbol connections, symbol-text qualifiers).

A.2.1. Input Adaptation

Input: d + pre-defined boxes $B_{\text{input}} = \{b^1, ..., b^N\}$ (N = number of entities). For each $b^k = [x^k_1, y^k_1, x^k_2, y^k_2]$:

1. Normalize b^k to the scale of x^* (Swin stage 4 feature map): $b^{k'} = b^k \times [W^*/W, H^*/H, W^*/W, H^*/H]$.
2. Extract region features via 7×7 RoIAlign: $f^{\text{ent}}_k = \text{RoIAlign}(x^*, b^{k'}, 7)$.

A.2.2. Hyper-Relation Prediction

DTRT models two core hyper-relations for engineering diagrams—symbol-symbol connection relations (solid/non-solid, 2 classes for P&IDs) and symbol-text qualifier relations (e.g., "valve" + "DN50")—both refined and generated by a 9-layer Triplet Decoder ($L_t = 9$):

1. Coupled Query Initialization: Merge learnable S/O queries with high-frequency triplet semantic embedding: $T_{\text{init}} = [S_{\text{learn}} + S_e, O_{\text{learn}} + O_e]$.
2. CSA: $[S^{\text{csa}}_m, O^{\text{csa}}_m] = \text{msa}(Q = K = V = T_m) + T_m$ (avoids redundant pairs).
3. DDVA: Locate S/O centers $[b^m_s, b^m_o]$, adjust sampling points:$[S^{\text{ddva}}_m, O^{\text{ddva}}_m] = \delta\text{-}a([S^{\text{csa}}_m, O^{\text{csa}}_m], x^*, \Delta p_b + \lambda_p p_m)$ ($\Delta p_b = 0.8$, $\lambda_p = 0.6$).
4. Relation Generation: Project refined T_{L_t} to probabilities: $\hat{r} = \text{Softmax}(\text{MLP}_{\text{rel}}(T_{L_t}))$;

A.3. Scene Graph Detection

SGDET generates end-to-end ¡box, category, relation¿ triplets from raw images, unifying Entity Detection and relation prediction (90.5 GFLOPs).

A.3.1. End-to-End Pipeline

The pipeline shares the dynamic token backbone (Sec. A.1.2) to extract pruned feature x^*, then processes x^* via two decoupled but synergistic branches—Entity Branch and Relation Branch—where entity queries (from Entity Branch) do not directly compute with triplet queries (from Relation Branch), but serve as spatial priors to indirectly guide triplet refinement:

1. **Entity Branch: Generate Spatial Priors**
   Uses a 6-layer Entity Decoder ($L_e = 6$) to output refined entity queries E_{L_e} (spatial/semantic information) as priors:
   Query initialization: MQS combines high-confidence candidates and learnable queries Θ_e;
   CDN optimization: Guides query clustering via Y^+ (small Gaussian noise) and Y^- (large Gaussian noise), loss $\mathcal{L}_{\text{dn}}$;
   Output: E_{L_e} (stored as spatial prior, no direct participation in triplet computation).
2. **Relation Branch: Triplet Refinement with Prior**
   Uses a 9-layer Triplet Decoder ($L_t = 9$) to generate hyper-relations, with triplet queries refined indirectly by E_{L_e}:
   Query initialization: Coupled S/O queries merged with high-frequency triplet embeddings: $T_{\text{init}} = [S_{\text{learn}} + S_e, O_{\text{learn}} + O_e]$;
   CSA: Models S/O dependency to avoid redundancy: $[S_m^{\text{csa}}, O_m^{\text{csa}}] = \text{msa}(Q = K = V = T_m) + T_m$;
   DDVA: Captures visual hints (e.g., lines) via adjusted sampling points;
   DEA: Indirectly guides refinement by treating E_{L_e} as attention key/value (no direct feature fusion), avoiding $O(N^2)$ complexity:$T_{m+1} = \text{msa}(T_m + T_e, E_{L_e}, E_{L_e}) + T_m$.

A.3.2. HKG Generation & Loss

1. HKG Construction: Project T_{L_t} to triplets $\hat{p} = \{\hat{s}, \hat{r}, \hat{o}\}$, extract qualifiers $\hat{q}$ via relation categories, assemble $\{\hat{s}, \hat{r}, \hat{o}, \hat{q}\}$ with PaddleOCR.
2. Total Loss: $\mathcal{L}_{\text{SGDET}} = \mathcal{L}_{\text{ent}} + \mathcal{L}_{\text{dn}} + \lambda_{\text{rel}}\mathcal{L}_{\text{CE}}(\hat{r}, Y_{\text{rel}}) + \lambda_{\text{sync}}\mathcal{L}_{\text{sync}}$ ($\lambda_{\text{rel}} = 1.0$, $\lambda_{\text{sync}} = 0.1$);
3. Inference Pruning: Retain high-confidence entities (confidence>0.5) and triplets (confidence>0.3).

A.4. Pseudocode

The whole procedure of DTRT generating HKGs is shown as Algorithm 1.

Algorithm 1 Pseudocode for DTRT

Require: Raw diagram: d, Ground-truth for entities and relations: $Y = \{Y_{\text{cls}}, Y_{\text{bbox}}, Y_{\text{rel}}\}$,
Hyperparams: $\rho_r = 0.7$, $L_e = 6$, $L_t = 9$, $\lambda_e = 1.0$, $\lambda_{\text{rc}} = 0.5$, $\lambda_{\text{dn}} = 0.3$, $\lambda_{\text{sync}} = 0.1$
Ensure: Final output: HKG, Total loss: Loss

```
function DTRT(d, Y)
    M ← InitDTRTModel(ρ_r, L_e, L_t)
    optimizer ← AdamW(M.parameters(), lr = lr)
    d_prep ← pre-process(d)
    total_HKG, total_L ← [], 0.0
    optimizer.zero_grad()
    function DynamicTokenBackbone(d_p)
        x*, L_rc ← DT(swin(d_p), ρ_r), mse(RC(x*, d_p), d_p)
        return x*, L_rc
    end function
    function EntityBranch(x_star, Y_cls, Y_bbox)
        E ← ent_dec(mixed_q(x_star), x_star, L_e)
        b̂ ← bbox_h(E), ĉ ← cls_h(E)
        L_ent ← λ_e · (CE(ĉ, Y_cls) + L1+GIoU(b̂, Y_bbox))
        L_dn ← λ_dn · CE(ĉ, cdn(Y))
        return E, ĉ, b̂, L_ent, L_dn
    end function
    function RelationBranch(x_star, E, Y_rel, ĉ)
        T ← triplet_dec(coupled_q(), x_star, E, L_t)
        r̂ ← rel_h(T)
        L_rel ← λ_rel · CE(r̂, Y_rel)
        L_sync ← λ_sync · conf_align(ĉ, r̂)
        return r̂, L_rel, L_sync
    end function
    function GenerateHKG(ĉ, b̂, r̂, d_p)
        high_ent ← filter(ĉ, b̂, 0.5)
        high_rel ← filter(r̂, 0.3)
        quals ← ocr_match(d_p, high_ent)
        return assemble(high_ent, high_rel, quals)
    end function
    for d_p ∈ d_prep do
        (x*, L_rc) ← DynamicTokenBackbone(d_p)
        (E, ĉ, b̂, L_ent, L_dn) ← EntityBranch(x*, Y_cls, Y_bbox)
        (r̂, L_rel, L_sync) ← RelationBranch(x*, E, Y_rel, ĉ)
        HKG.extend(GenerateHKG(ĉ, b̂, r̂, d_p))
        Loss+ = 1/|d_prep| · (L_ent + λ_rc L_rc + L_dn + L_rel + L_sync)
    end for
    Loss.backward()
    optimizer.step()
    lr_scheduler.step()
    return HKG, M
end function
HKG = DTRT(d, Y)
```

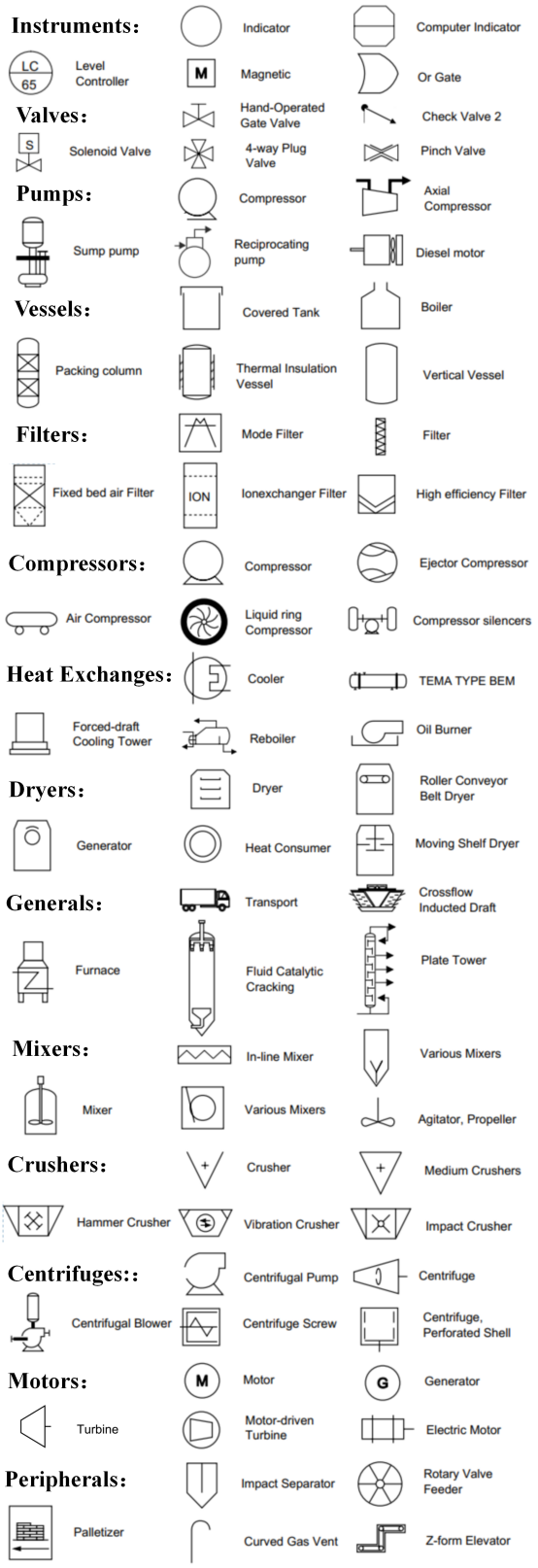

Figure 4. Symbol categories of P&ID dataset

B. Experiment Details

B.1. Data Details

B.1.1. P&ID Dataset

The P&ID dataset was constructed by integrating and refining two publicly available datasets, with additional annotations to fill gaps in relational and textual information:

1. Digitize-PID: A synthetic P&ID dataset that provides basic image data and initial entity annotations (e.g., symbols for valves, instruments, and pipes).
2. PID2Graph: An extended dataset built on Digitize-PID, which increases data volume and adds preliminary relationship annotations between symbols.

To address the lack of text-related annotations and incomplete hyper-relational labels in the above datasets, we further integrated their image and annotation resources, supplemented text entity annotations, and added relation annotations between symbols and their corresponding text labels.

The constructed P&ID dataset contains 762 8K images, consistent with the typical resolution of industrial P&IDs to ensure the model is evaluated under real-world conditions. The dataset includes three levels of annotations to support hyper-relational knowledge graph (HKG) generation:

1. Entity Annotations (16 categories): include instrument, valves, pumps, vessels, filters, compressors, heat exchanges, dryers, general, mixers, crushers, centrifuges, motors, peripheral, and piping&connection symbols.
2. Relation Annotations: Two types of relational annotations are provided to model topological and attribute connections, which are connection relations (solid connection and non-solid connection) and qualifier relations (symbol qualifiers and connection qualifiers).
3. Text Annotations: Text entities (e.g., component model numbers, parameters, and labels) are annotated with bounding boxes and content. These text annotations are linked to their corresponding symbols to serve as qualifiers in the HKG, enabling the model to associate structural knowledge with textual attributes.

B.1.2. ED Dataset

The ED dataset was compiled by merging multiple circuit diagram resources and standardizing annotations to ensure consistency. Integrating the ED dataset from AMSNet[37], the hand-drawn circuit diagram dataset from [38] and online unlabeled circuit diagrams, the final ED dataset contains 4768 images with appended annotations.

Similar to the P&ID dataset, ED annotations are designed to support end-to-end hyper-relational extraction:

1. Entity Annotations (12 categories): include resistors, capacitors, inductors, diodes, transistors, integrated circuits (ICs), switches, groundings, crossings, bridges, power supplies and general symbols.

2. Relation Annotations: connection relations and qualifier relations.
3. Text Annotations.

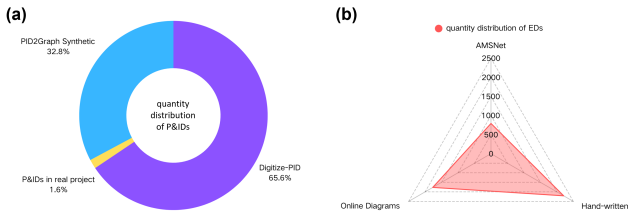

Figure 5. The distribution of the number of images in P&ID and ED datasets.

B.1.3. Key Contributions of the Datasets

Both datasets address critical limitations of existing engineering diagram datasets and provide unique value for research:

1. Hyper-Relational Annotations: Beyond basic entity and relation labels, they include qualifier annotations to model complex attribute relationships, supporting HKG generation.
2. Text-Symbol Alignment: Explicitly link text to symbols, enabling models to extract both structural and textual knowledge— a key requirement for industrial digitization.
3. Standardized Evaluation: Enable fair comparison of models on entity detection (AP/AR) and scene graph generation (R@K) tasks, advancing research in engineering diagram understanding.

B.2. Supplementary experiments

B.2.1. Single Task v.s. Multi-Task

This experiment compares the performance of single-task training (isolated entity detection, isolated relation extraction) and multi-task training (end-to-end entity detection + relation extraction + hyper-relational knowledge graph (HKG) generation) to verify the synergy between tasks in DTRT. The experiment is setup under the following configuration:

1. Single-Task Configurations: In single-entity task, we only train the entity decoder to optimize entity detection loss ($\mathcal{L}_{ent} + \mathcal{L}_{dn}$). In single-relation task, we fix pre-trained entity features (from Single-Entity) and only train the triplet decoder to optimize relation loss ($\mathcal{L}_{rel} + \mathcal{L}_{sync}$).
2. Multi-Task Configuration: Jointly train all modules (dynamic token backbone + entity decoder + triplet decoder) with the total loss$\mathcal{L} = \lambda_{rc}\mathcal{L}_{rc} + \lambda_{e}\mathcal{L}_{ent} + \lambda_{dn}\mathcal{L}_{dn} + \lambda_{rel}\mathcal{L}_{rel} + \lambda_{sync}\mathcal{L}_{sync}$.

Table 7 summarizes the performance on both datasets. Multi-task training outperforms single-task settings in all metrics, as the mutual guidance between entity detection and relation extraction enhances feature representation: On P&ID, multi-task training achieves 1.2% higher AP_{50} and 3.5% higher R@1000 than single-task counterparts, thanks to the spatial priors from entity queries guiding relation modeling. On ED, the synergy is more pronounced (1.5% higher AR_{50}, 4.1% higher R@200) due to the complex connection topology, where accurate entity localization directly improves relation inference. Computational efficiency (GFLOPs) is also improved by 8–12% in multi-task mode, as shared feature extraction (via the dynamic token backbone) avoids redundant computations across tasks.

B.2.2. ResNet v.s. ViT

This experiment compares two mainstream visual backbones—ResNet-50 (convolutional architecture) and Swin Transformer-Tiny (Vision Transformer, ViT-based, used in original DTRT)—to validate the advantage of ViT in modeling global relational features for engineering diagrams. The experiment is conducted under the following configuration:

1. ResNet-50: Replace the dynamic token backbone with ResNet-50 (fine-tuned on ImageNet).
2. Swin-T (ViT): Original DTRT backbone, with 4-stage shifted windows and dynamic token pruning.

For other Modules, we keep the entity decoder ($L_e = 6$) and triplet decoder ($L_t = 9$) identical to isolate the backbone impact. Table 8 shows that ViT-based backbones outperform ResNet in relational tasks, while maintaining competitive entity detection performance. On P&ID, Swin-T achieves 6.63% higher R@1000 than ResNet-50; on ED, the gap expands to 8.17%. This is because ViT's global attention better captures long-range dependencies (e.g., P&ID pipe connections, ED wire topologies) compared to ResNet's local convolution. Both backbones perform similarly on AP_{50} (within 0.5%), but Swin-T achieves 1.2–1.5% higher AR_{50}, as dynamic token pruning preserves more small-scale entities (e.g., ED diodes) than ResNet's fixed feature maps. Swin-T reduces GFLOPs by $\sim$34% compared to ResNet-50, thanks to token pruning (ResNet-50's pruned version still retains more redundant features due to convolutional inductive bias).

B.2.3. Pruning More v.s. Pruning Less

The dynamic token pruning module in DTRT uses a default pruning rate $\rho_r = 0.7$ (pruning 70% low-value tokens). This experiment explores the impact of pruning rate ($\rho_r \in 0.5, 0.7, 0.9$) on performance and computational efficiency. The experiment is setup under the following configuration:

1. $\rho_r = 0.5$: Prune 50% tokens (retain more features, higher computation).
2. $\rho_r = 0.7$: Default, prune 70% tokens.
3. $\rho_r = 0.9$: Prune 90% tokens (retain minimal features,

lower computation).

We measure entity detection (AP_{50}/AR_{50}), relation extraction (R@1000/R@200), and GFLOPs on both datasets. Table 9 reveals a trade-off between performance and efficiency, with $\rho_r = 0.7$ achieving the optimal balance:

1. Pruning Less ($\rho_r = 0.5$): GFLOPs increase by 30% compared to $\rho_r = 0.7$ (P&ID: 127.8 vs. 90.5 GFLOPs) but only improves AP50 by 0.2–0.3% and R@1000 by 0.5–0.7%. Redundant tokens (e.g., blank regions in P&IDs) add computation without enhancing feature quality.
2. Pruning More ($\rho_r = 0.9$): GFLOPs decrease by 20% (P&ID: 72.3 GFLOPs) but causes significant performance drops: AP50 falls by 2.89% (P&ID) and 3.12% (ED), R@1000 drops by 8.45% (P&ID)—critical features (e.g., ED wire segments, P&ID text labels) are pruned.

At the end, we balance the efficiency and performance, retaining 30% high-value tokens (focused on symbols, lines, and texts) while reducing GFLOPs by 69% compared to no pruning (Table 2 in main text).

B.2.4. More Visualization of Results

This section supplements qualitative visualizations to illustrate DTRT's performance in entity detection, relation extraction, and token pruning. All visualizations use representative samples from the P&ID and ED datasets.

As shwon in Figure 6, DTRT successfully detected the symbol and text entities and predicted the connection relation and qualifier relation. The detection results are shown in boxes of different colors, the connection relation is visualized as red arrows.

As shown in Figure 7, the proposed dynamic token pruning mechanism is capable of filtering unrelated tokens and preserve high value visual information. This successfully reduced model computation, while preserving detailed image features.

Table 7. Performance Comparison: Single Task vs. Multi-Task

Setting	**Dataset**	$\mathbf{AP_{50}}$ (↑)	$\mathbf{AR_{50}}$ (↑)	**R@1000/R@200** (↑)	**GFLOPs** (↓)
Single-Entity	P&ID	97.81%	95.84%	-	102.3
	ED	98.26%	96.03%	-	98.7
Single-Relation	P&ID	-	-	91.34%	89.6
	ED	-	-	88.42%	85.2
Multi-Task (DTRT)	P&ID	99.01%	97.04%	94.84%	90.5
	ED	99.76%	97.53%	92.52%	87.3

Table 8. Performance Comparison: ResNet-50 v.s. Swin Transformer-Tiny

Backbone	**Dataset**	$\mathbf{AP_{50}}$ (↑)	$\mathbf{AR_{50}}$ (↑)	**R@1000/R@200** (↑)	**GFLOPs** (↓)
ResNet-50	P&ID	98.53%	95.84%	88.21%	136.8
	ED	99.27%	96.33%	84.35%	132.5
Swin-T (pruned)	P&ID	99.01%	97.04%	94.84%	90.5
	ED	99.76%	97.53%	92.52%	87.3

Table 9. Performance Comparison: Different Token Pruning Rates

Pruning Rate (ρ_r)	**Dataset**	$\mathbf{AP_{50}}$ (↑)	$\mathbf{AR_{50}}$ (↑)	**R@1000/R@200** (↑)	**GFLOPs** (↓)
0.5 (Prune 50%)	P&ID	99.23%	97.31%	95.38%	127.8
	ED	99.98%	97.86%	93.15%	122.6
0.7 (Prune 70%, Default)	P&ID	99.01%	97.04%	94.84%	90.5
	ED	99.76%	97.53%	92.52%	87.3
0.9 (Prune 90%)	P&ID	96.12%	94.21%	86.39%	72.3
	ED	96.64%	94.41%	84.07%	69.8

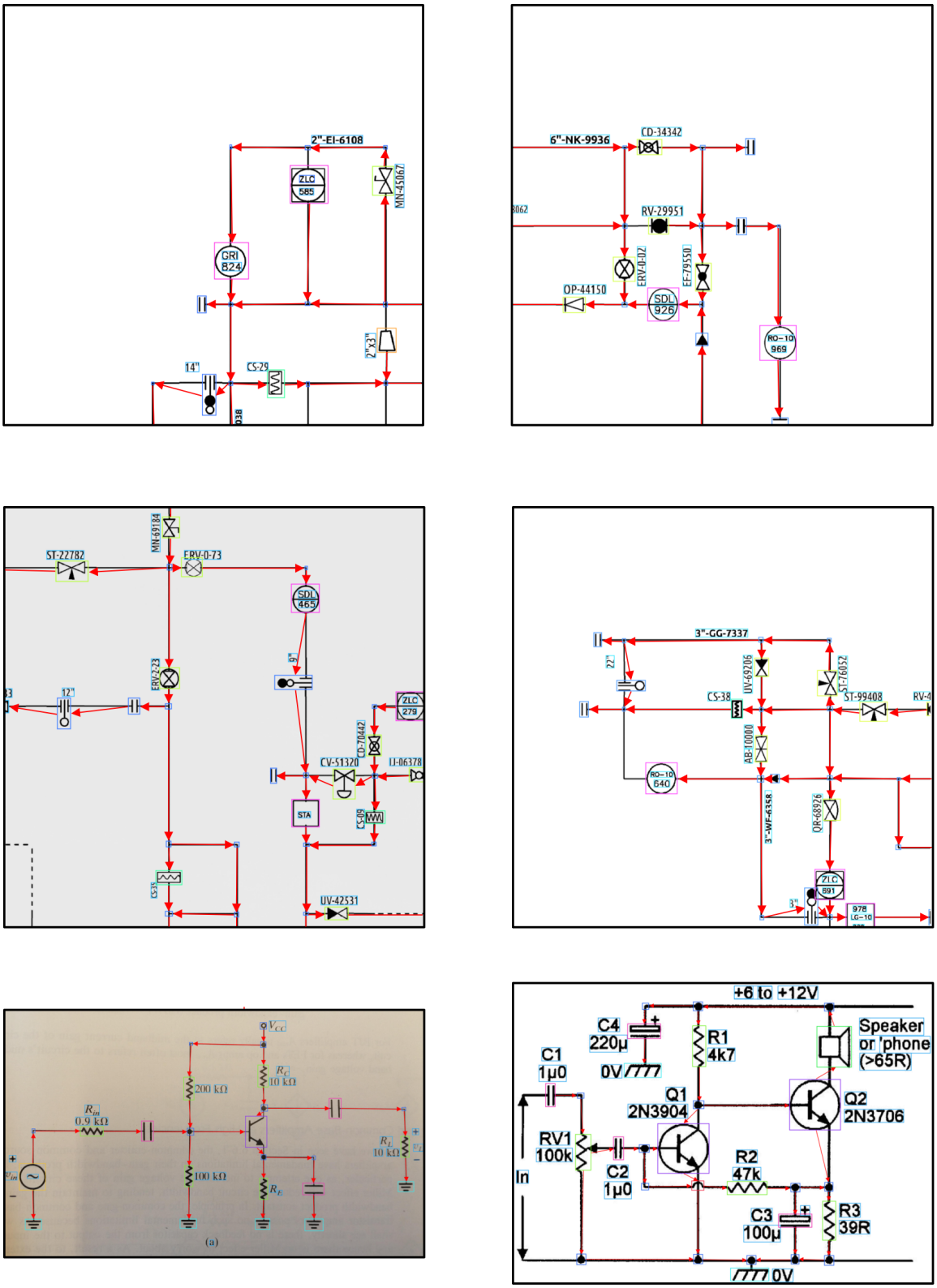

Figure 6. The visualization of DTRT outputs.

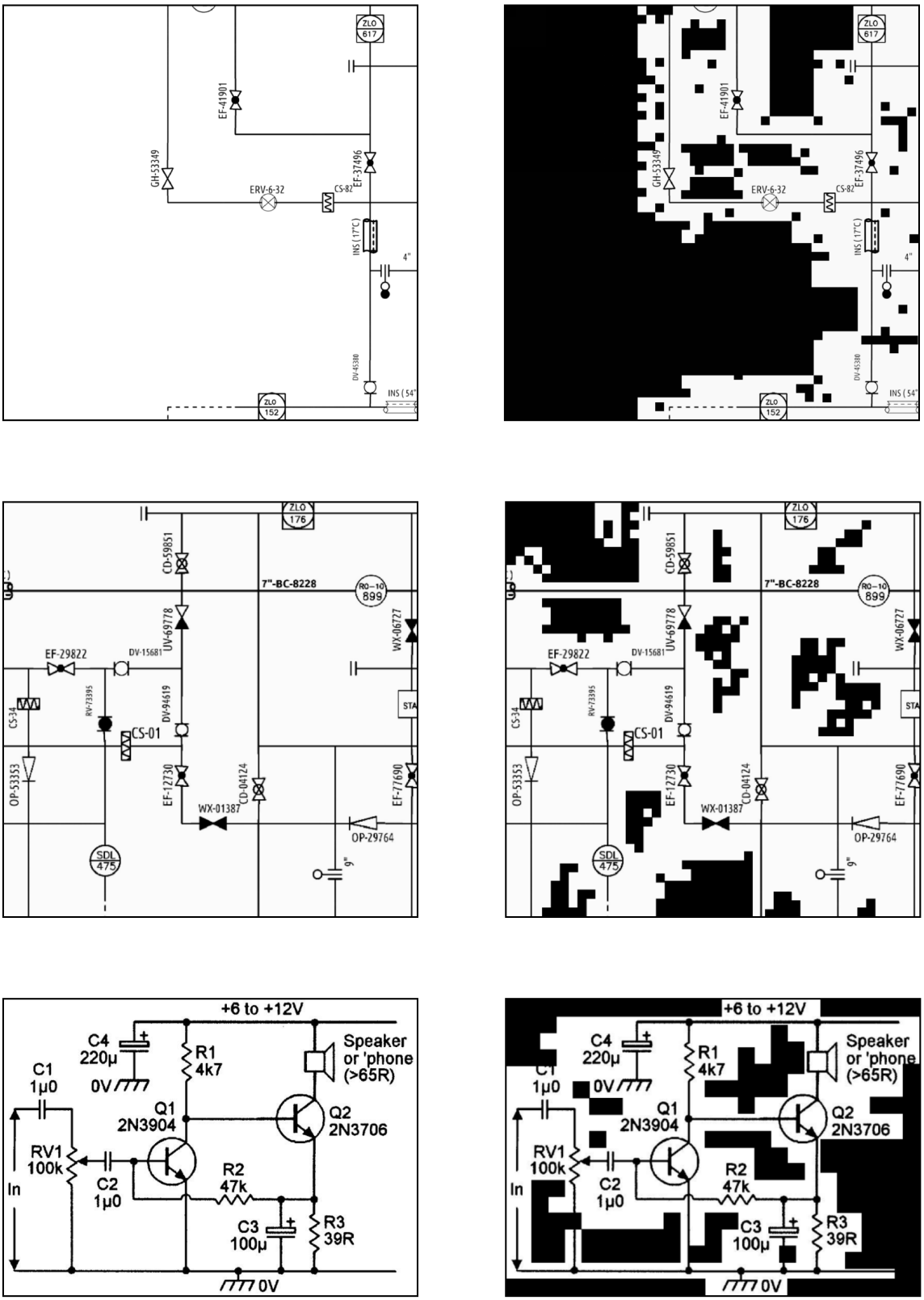

Figure 7. The visualization of dynamic token pruning.