A Target-Aware Neural Network Inference Template Code Generation Technique for the TANGO Framework

Jaebok Park
On-Device Artificial Intelligence
Models Research Section
ETRI
Daejeon, Korea
parkjb@etri.re.kr

Kyunghee Lee
On-Device Artificial Intelligence
Models Research Section
ETRI
Daejeon, Korea
kyunghee@etri.re.kr

Changsik Cho
On-Device Artificial Intelligence
Models Research Section
ETRI
Daejeon, Korea
cscho@etri.re.k

***Abstract*—Artificial intelligence-based services consist of a complex workflow that includes generating and optimizing neural network models, creating application-specific template code, and deploying the models on target devices. An integrated and automated framework is needed to systematically handle the entire pipeline. However, these processes demand a high level of expertise and technical skill, posing a significant burden for general developers. In particular, additional optimization is necessary to enable neural network inference tailored to diverse target environments. This paper proposes an automated method that allows users to generate neural networks optimized for selected target devices and application services, and to package them into executable inference code for final deployment—without requiring complex configuration. Specifically, the proposed template code generation technique for neural network inference is based on a predefined template skeleton. This skeleton is designed with consideration for a wide range of hardware accelerators, allowing for general applicability across diverse environments. The proposed approach enables the generation of final executable code with minimal modifications by adapting to the computational resources and accelerator characteristics of each target device.**

Keywords— TANGO, AutoML, Deep, Framework, Template

I. INTRODUCTION

In recent years, the application of neural network–based artificial intelligence technologies has been expanding across various industrial sectors. However, the development of such technologies still demands a high level of expertise. In particular, small and medium-sized enterprises often struggle to secure professionals capable of designing neural network models and integrating them into application services for deployment. For general software developers, managing the entire process of neural network development and deployment poses significant technical challenges. Therefore, there is a growing demand for automated tools or supportive frameworks that enable non-experts to easily develop and deploy neural network-based application services [1].

The entire process from neural network model creation to deployment is consisted of complex and repetitive tasks, and manual approaches often face limitations in terms of productivity and consistency. As a result, the need for MLOps frameworks [2][3], which support automation and systematic management of model development and operations, has become increasingly prominent. MLOps facilitates practical and scalable deployment of neural network–based systems by supporting the full pipeline from rapid model development to reliable deployment and continuous performance monitoring.

This paper proposes a technique, implemented using the TANGO framework developed by ETRI [4], for optimizing and deploying neural networks according to the performance characteristics of five distinct target devices. TANGO (Target Aware No-code Neural Network Generation and Operation Framework) is an MLOps system that enables the generation and deployment of such target-specific neural networks through a no-code approach.

TANGO first allows users to specify requirements through a configuration wizard, where target devices and neural network applications can be selected easily. Based on this specification, a suitable neural network model is then generated. The model generation process utilizes NNI [5] along with TANGO's recommendation system to automatically produce neural networks optimized for the selected target device and application service. The TANGO recommendation system suggests optimal neural network models for image classification and object detection tasks, depending on the performance capabilities of the target device. Target devices are categorized into performance tiers. For image classification, the system recommends five models—ResNet203, 152, 101, 50, and 34 [6]; for object detection, it recommends six YOLOv9 models—T, S, M, C, and E [7]. Neural network model recommendation accumulates experiential data as the number of training iterations increases. By training the recommendation model on this data, it becomes possible to implement a more efficient recommendation algorithm.

Next, the generated neural network is converted into a format compatible with the inference engine of the target device, and optimizations such as quantization and pruning are applied. The proposed template code generation method defines a skeleton for neural network inference templates based on nine essential code components. This method is generalized to accommodate various inference engines used across different target devices, thereby providing a standardized approach for generating executable template code.

This paper presents an automated template generation method to support efficient neural network inference on various target devices. The TANGO framework generates and deploys both the optimized model and the corresponding executable code to the target device, enabling rapid and seamless deployment. The proposed method provides a template generation technique in TANGO that automatically searches, trains, optimizes, and deploys neural networks based on the target device and application requirements.

II. Neural Network Inference Template Generation Method Based on the TANGO

TANGO, developed by ETRI, allows non-expert users to easily develop neural network–based applications with little to no coding. It is designed as an automated framework for neural network generation and deployment that supports automatic distribution to target devices. TANGO can be largely divided into neural network generation and deployment. First, TANGO's neural network generation proceeds through the process of generating a neural network by selecting basic requirements such as the target device and tasks like Detection/Classification, as provided in Table 1.

TABLE I. Device Adaptive Neural Network Model Recommendation

Target device		Model recommendation	
grade	detail spec.	Obect detection	Image Classifcation
PC	PC_Server	Yolov7_E6	Resnet152
	PC	Yolov7_W6	Resnet152
On Device	Jetson AGX Orin	Yolov7_W6	Resnet101
	Jetson AGX Xavier	Yolov7_X	Resnet50
	Galaxy S22	Yolov7_Tiny	Resnet34
	Rasberry Pi5	Yolov7_Tiny	Resnet34
	Odroid-N2	Yolov7_Tiny	Resnet34
	Odroid-M1	Yolov7_Tiny	Resnet34

The end-to-end TANGO workspace automates the machine learning pipeline, including data preprocessing, model selection, hyperparameter tuning, and model deployment. The neural network model generation process begins by selecting a base model derived from State-of-the-Art (SOTA) architectures. Next, Neural Architecture Search (NAS) and Hyperparameter Optimization (HPO) techniques are employed to automatically explore the optimal model structure and configuration, followed by a retraining process. As shown in Figure 1, TANGO automatically trains a neural network according to the desired requirements by selecting a suitable model optimized for the target device chosen by the user.

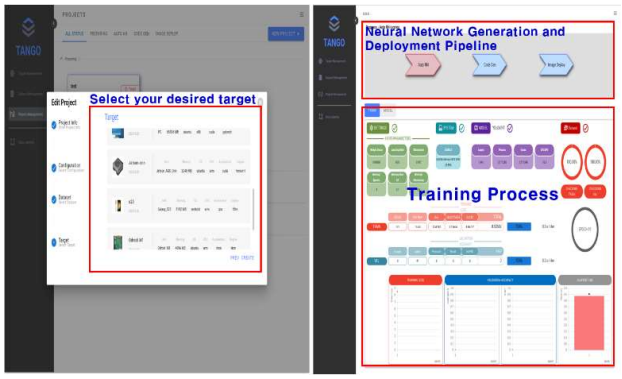

Fig. 1. A target selection and training process of TANGO

As shown in Figure 2, TANGO generates an executable file specific to the target environment based on the user-defined neural_net_info.yaml and deployment.yaml, including the necessary libraries and pre/post-processing code. Next, this is compressed or built as a container image and installed and deployed to the device. TANGO deployment supports the automatic generation of executable code, environment optimization, accelerator support, inference engine support, and deployment convenience.

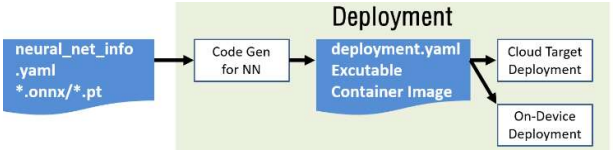

Fig. 2. TANGO's distribution module configuration diagram

The template code generation technique fills in the implementation of the detect function according to the YAML configuration and the characteristics of the target device, as illustrated in Figure 3. First, the acceleration environment for the target device is initialized. Then, the neural network model generated by the model generation module is loaded, and a dataloader is configured to supply input images or videos for inference. Next, input images are retrieved as a stream within a for loop, and inference is performed on each image. Since the inference code depends on the supported data types and input dimensions of each target device, additional modifications are required to generate device-specific executable code. Non-Maximum Suppression (NMS) is applied to eliminate redundant bounding boxes, retaining only the one with the highest confidence score. Subsequently, detection results are processed and bounding boxes are drawn around the detected objects. The resulting images are then output as a stream. TANGO's template code generation utilizes a basic template skeleton that is independent of accelerator type. The final template code is produced by selectively modifying only the portions required for each target device.

```
def detect(save_img=False):
    # Set device (CPU/GPU/NPU/Adreno/TPU/Mali)
    device = select_device(opt.device)   1: Set device

    # Load model   2: Load model
    model = attempt_load(weights, map_location=device)  # load FP32 model

    # Set Dataloader   3: Set dataloader
    dataset = LoadImages(source, img_size=imgsz, stride=stride)

    for path, img, im0s, vid_cap in dataset:   4: Load image
        img = torch.from_numpy(img).to(device)
        # Inference   5: Inferecnce
        with torch.no_grad():
            pred = model(img, augment=opt.augment)[0]

        # Apply NMS   6: Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, nms)

        # Process detections   7: Process detection
        for i, det in enumerate(pred):  # detections per image
            if webcam:  # batch_size >= 1
                p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
            else:
                p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)
                # Write results
                for *xyxy, conf, cls in reversed(det):   8: Process box
                    if save_img or view_img:  # Add bbox to image
                        label = f'{names[int(cls)]} {conf:.2f}'
                        plot_one_box(xyxy, im0, label=label, color=colors[int(cls)],
                                                                line_thickness=1)
            # Stream results   9: Output
            if view_img:
                cv2.imshow(str(p), im0)
```

Fig. 3. A skeleton of TANGO template code

TANGO's deployment is evaluated on five types of target devices, as illustrated in Figure 4. First, in the CPU-based acceleration environment, an optimized neural network is generated through quantization and pruning, taking into account the constraints of CPU-only execution. Subsequently,

template code is generated to execute the optimized neural network in accordance with the domain-specific service requirements. In the GPU-based acceleration environment, a neural network of appropriate size for Jetson ORIN is quantized and further optimized through TensorRT conversion. Additionally, template code is generated by integrating TensorRT processing into the predefined skeleton. The RKNN-based acceleration environment supports Rockchip NPU acceleration on the Odroid-M1. To enable neural network acceleration, the generated yoloE.pt model is first converted to the ONNX format and then optimized into the yoloE.rknn model through quantization and data type conversion. The template code is automatically modified by TANGO to incorporate RKNN processing logic. The TPU-based acceleration environment supports Google TPU acceleration on the Raspberry Pi. The neural network is converted to ONNX format and subsequently transformed into a TFLite model to support TPU-based execution. As with RKNN, the TANGO deployment module automatically updates the template code to include TPU-specific processing, generating the final executable program. The smartphone acceleration environment supports neural network execution using the Adreno 730 of the Galaxy S22 and the Adreno 740 of the Galaxy S23. The PyTorch-based neural network generated by TANGO is first converted to ONNX, then transformed into an OpenVINO-compatible format, and finally converted to a TFLite model to generate an Android-compatible YOLOv7 application. The final executable code is produced as an Android .apk package.

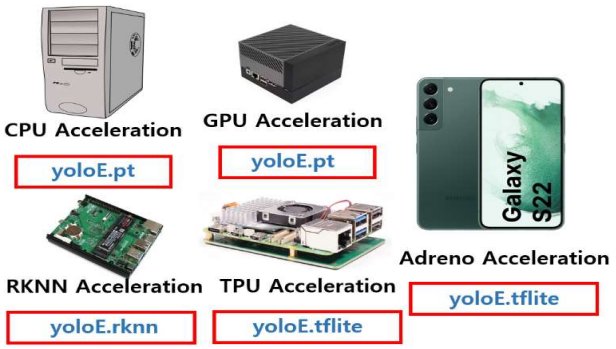

Fig. 4. Five target devices supported by TANGO

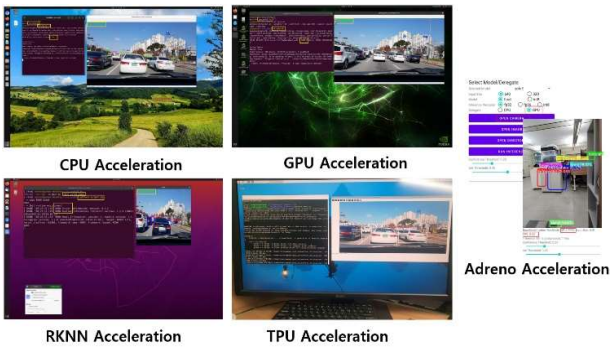

Fig. 5. Neural network execution screens for each device

Figure 5 shows the execution screens of the final neural networks and code running on each target device. In the CPU, GPU, RKNN, and TPU acceleration environments, real-time object recognition from video input is successfully demonstrated. On the smartphone, object recognition is performed directly through the live camera feed. These results demonstrate that TANGO automatically generates neural networks and execution code optimized for each of the five target devices.

III. CONCLUSION

This paper presents techniques for automatically generating neural networks and neural network application templates according to the target device. The proposed template code generation provides a template skeleton that can automatically generate inference code based on the characteristics of various target device accelerators using the generated neural network. Through this, developers with limited expertise in neural networks can utilize the neural network auto-generation function and the integrated development environment to develop AI services across various domains. As a result, it is expected that the base of neural network developers will expand, and the application scope of neural networks will broaden.

In the future, this research will be extended to technologies that automate application template code based on generative AI, particularly large language models (LLMs). Through this, LLMOps-based technologies will be developed to efficiently build generative AI application services. Ultimately, the goal is to develop general-purpose intelligent services that can be easily applied to various industrial fields, such as maritime, agriculture, and healthcare.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2021-0-00766, Development of Integrated Development Framework that supports Automatic Neural Network Generation and Deployment optimized for Runtime Environment and No. RS-2024-00397615, Development of an automotive software platform for Software-Defined-Vehicle (SDV) integrated with an AI framework required for intelligent vehicles)

REFERENCES

[1] Google, “Google AutoML Beta,” 2020. [Online]. Available: https://cloud. google.com/automl/, Accessed on: Jan. 24, 2020.

[2] Singh, K., Goswami, A., and Kukreja, S., “A Review on Interplay of AutoML and MLOps “AutoMLOps”: Current State, Challenges, and Future Scope.” Proc. - International Conference on Computing and Communication Networks (pp. 101-115). Springer, 2025.

[3] Alexander T., Difan D., Theresa E., Joseph G., Aditya M., Tim R., Sarah S., Daphne T., Tanja T., Henning W., and Marius L., “AutoML in the Age of Large Language Models: Current Challenges, Future Opportunities and Risks”. arXiv preprint arXiv:2306.08107, 2024.

[4] TANGO Project, https://github.com/ML-TANGO/TAN GO

[5] NNI(Neural Network Intelligence), https://github.com/microsoft/nni, Accessed in May, 2022.

[6] Kaiming H., Xiangyu Z., Shaoqing R., and Jian S., “Deep Residual Learning for Image Recognition”. arXiv preprint arXiv:1512.03385, 2015.

[7] Yolov9 Github, https://github.com/WongKinYiu/yolov9, Accessed in February, 2024.