TANGO

신경망 시각 프로그래밍 기술

성 명 이윤규 소 속 중앙대학교

2025 제4회 COMMUNITY CONFERENCE

TANGO

Target Adaptive No-code neural network Generation and Operation framework

주 관 ETRI 한국전자통신연구원 (TANGO)

주 최 과학기술정보통신부 IITP 정보통신기획평가원

문 의 parkjb@etri.re.kr / 042-860-5565

후 원 lablup weda tesla system (사)한국인공지능협회 SNUH 서울대학교병원 고려대학교 KOREA UNIVERSITY 홍익대학교 HONGIK UNIVERSITY CAU 중앙대학교 RTst Reliable & Trustworthy KETI 한국전자기술연구원 AIVN SUREDATA ACRYL 하늘소프트 TTA 한국정보통신기술협회 Telecommunications Technology Association

content

목 차

1 INTRO

1. Visual Programming
2. Visual Programming for Deep Learning

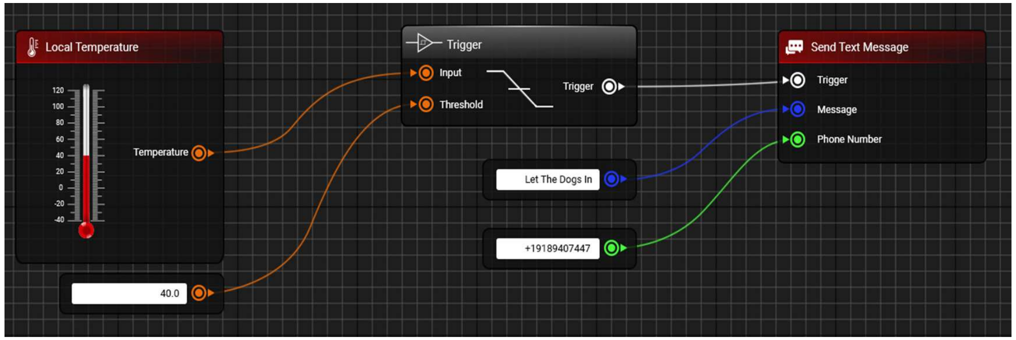
2 VELCRO

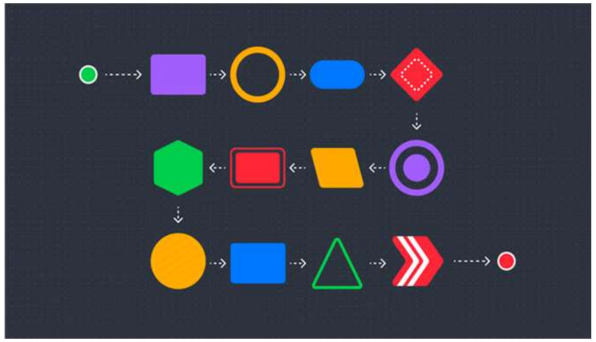
1. Proposed Method
2. Key Functionalities
3. Supported Architectures

3 VELCRO in TANGO

1. Workflow
2. Concluding Remarks

Visual Programming





- Enables users to program by manipulating graphical elements instead of text
- Operates on a 'No-Code' platform through an intuitive interface
- Utilizes graphical elements (e.g., icons, buttons, and symbols) in a coding format
- Explains operations and mechanisms in a way that even non-expert can understand

Visual Programming for Deep Learning

- Lack of user-friendly visualization and interface for complex architectures
- Insufficient automated code generation support in deep learning environments
- Limited capabilities for architecture editing, validation, and parameter updating
- Restricted diversity of supported frameworks, architectures, parameters, and layers

Name	Year	Type	Key Feature
DeepFlow	2025	Research tool	Flow-based visual design + debugging for DL.
DeepBlocks	2023	Research tool	Block-based visual DL modelling with complex branching support.
Hazel	2022	Open-source tool	Drag-and-drop to build, train and deploy DL via visual interface.
DL-IDE	2019	Research tool	No-code visual IDE for deep learning model design.
DeepVisual	2018	Research tool	Drag-and-drop layer components to design DL systems.

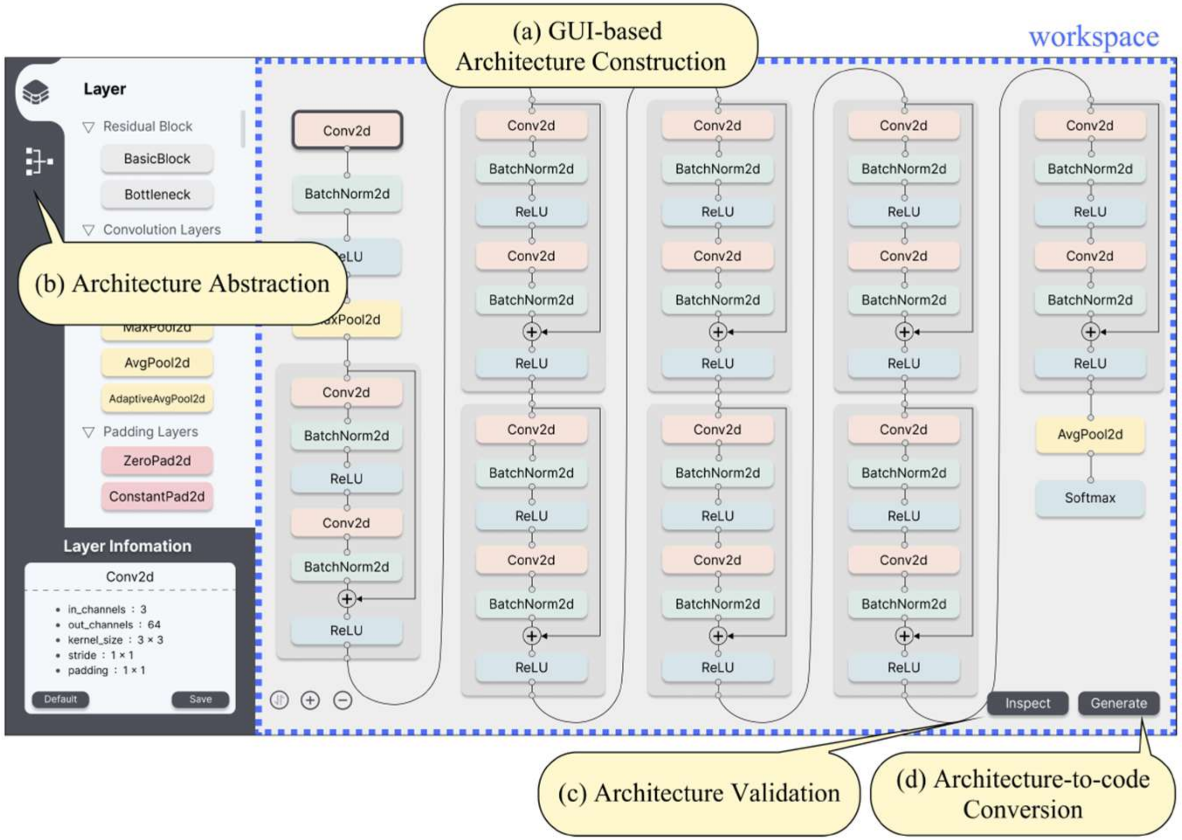
Proposed Method

VELCRO (Visual-based dEep Learning arChitecture geneRatOr)

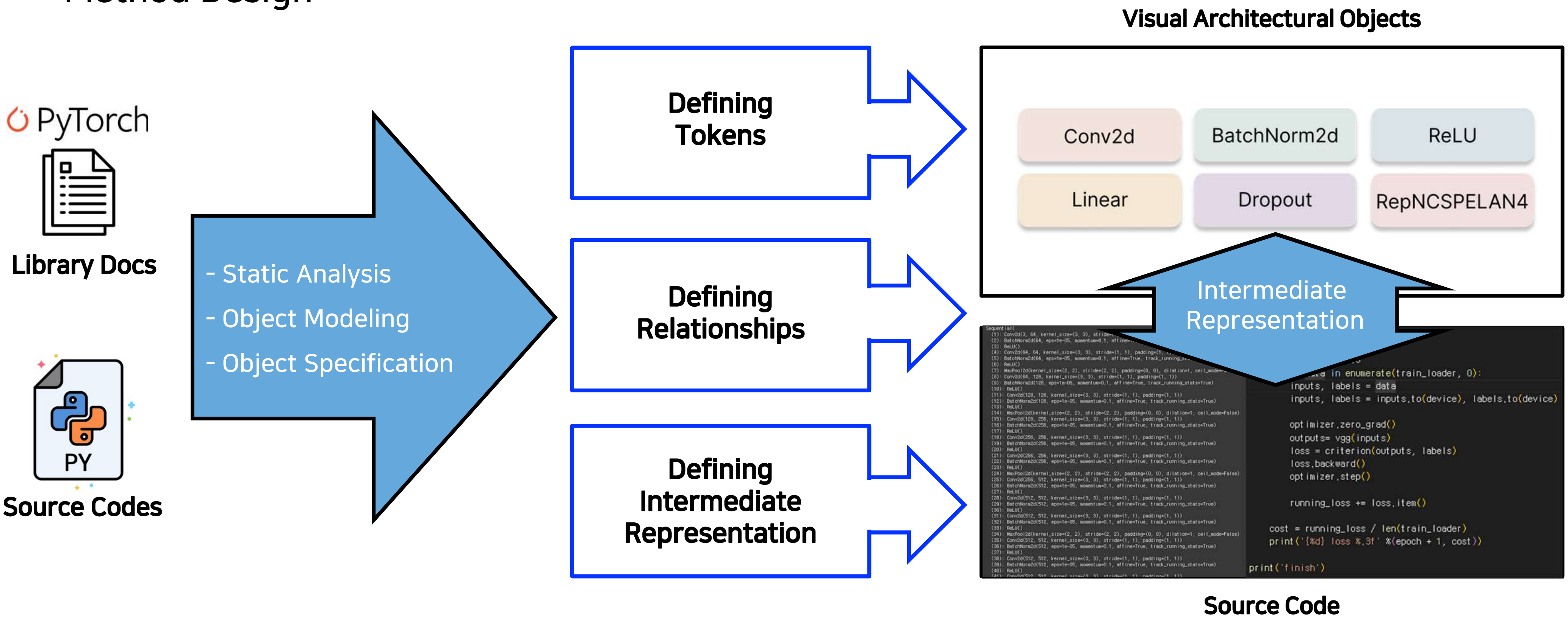
- User-friendly GUI for creation, validation, and modification of deep learning architectures
- Automation of precise deep learning code generation
- Accessibility to both expert and non-expert developers

Key Functionalities

- GUI-based Architecture Construction
- Architecture Abstraction
- Architecture Validation
- Architecture-to-code Conversion



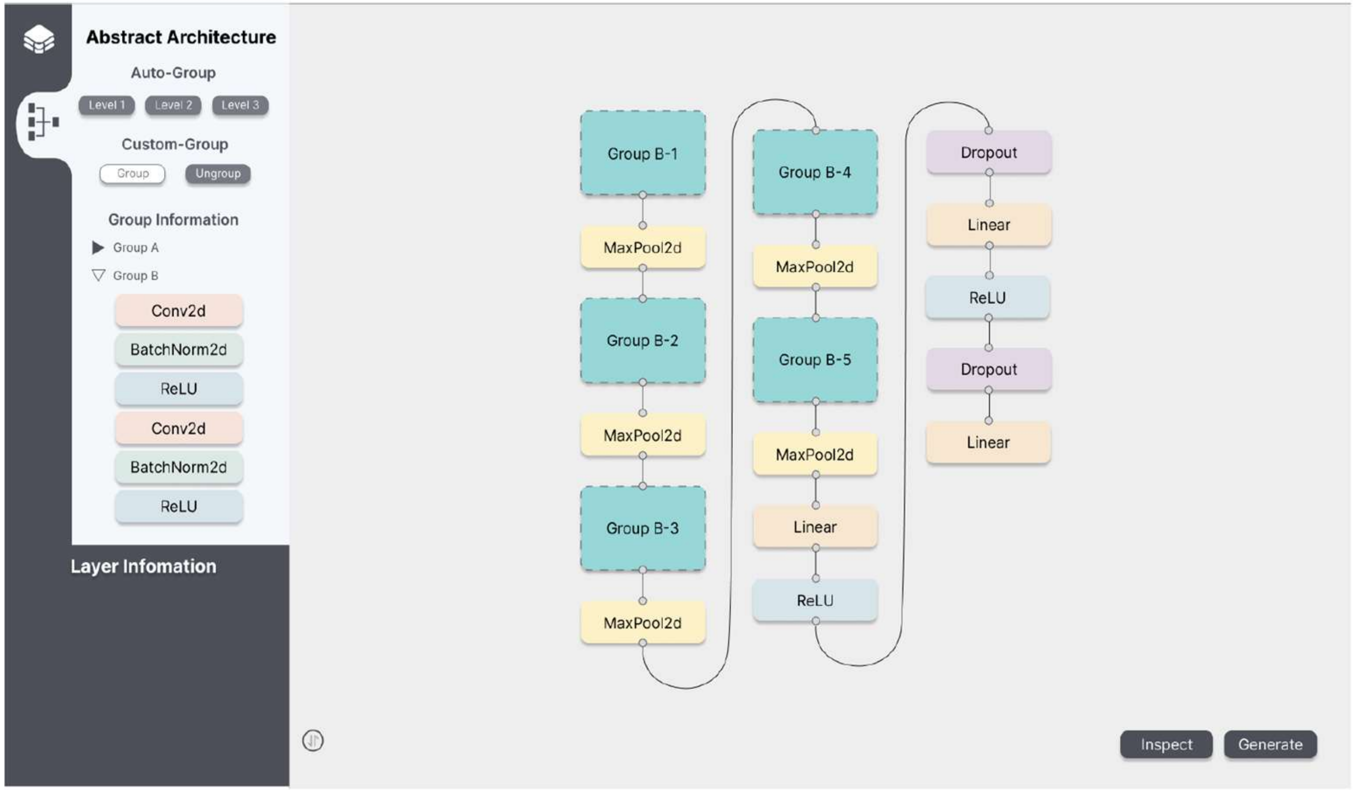
Method Design



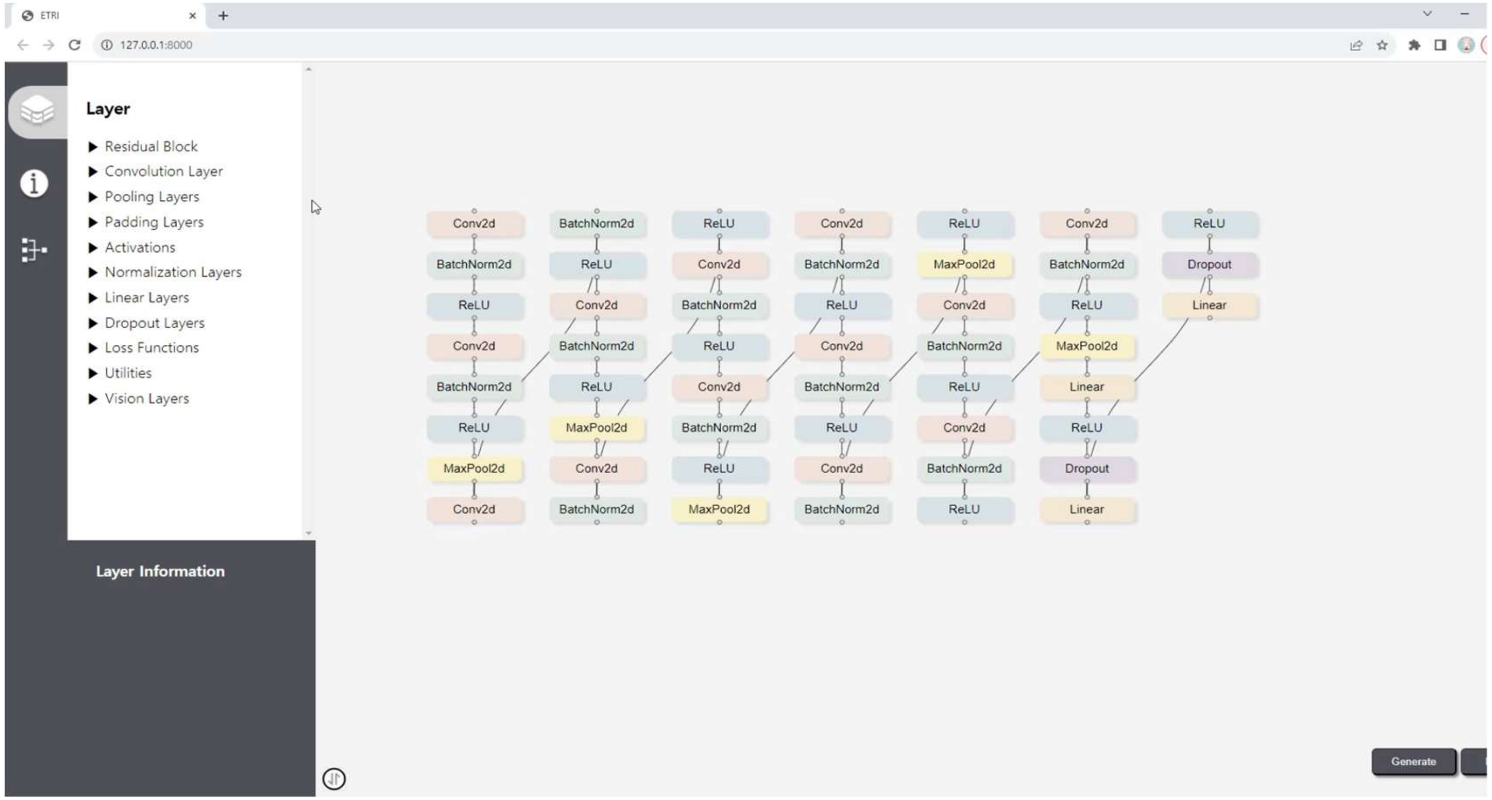
Key Functionalities: Architecture Abstraction

- Custom-group enables users have the flexibility to manually group selected nodes and edges into specific grouping levels based on their own criteria

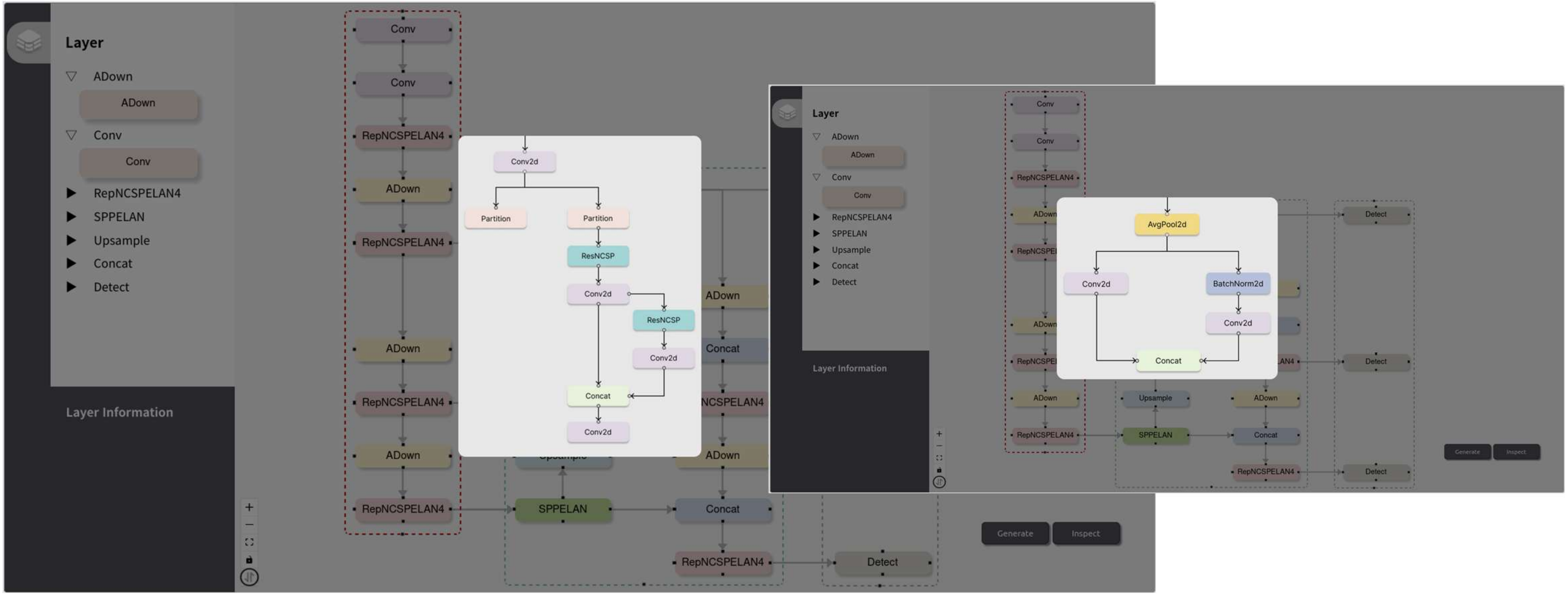
- Auto-group automatically organizes nodes and edges into predefined levels



Key Functionalities: Architecture Abstraction



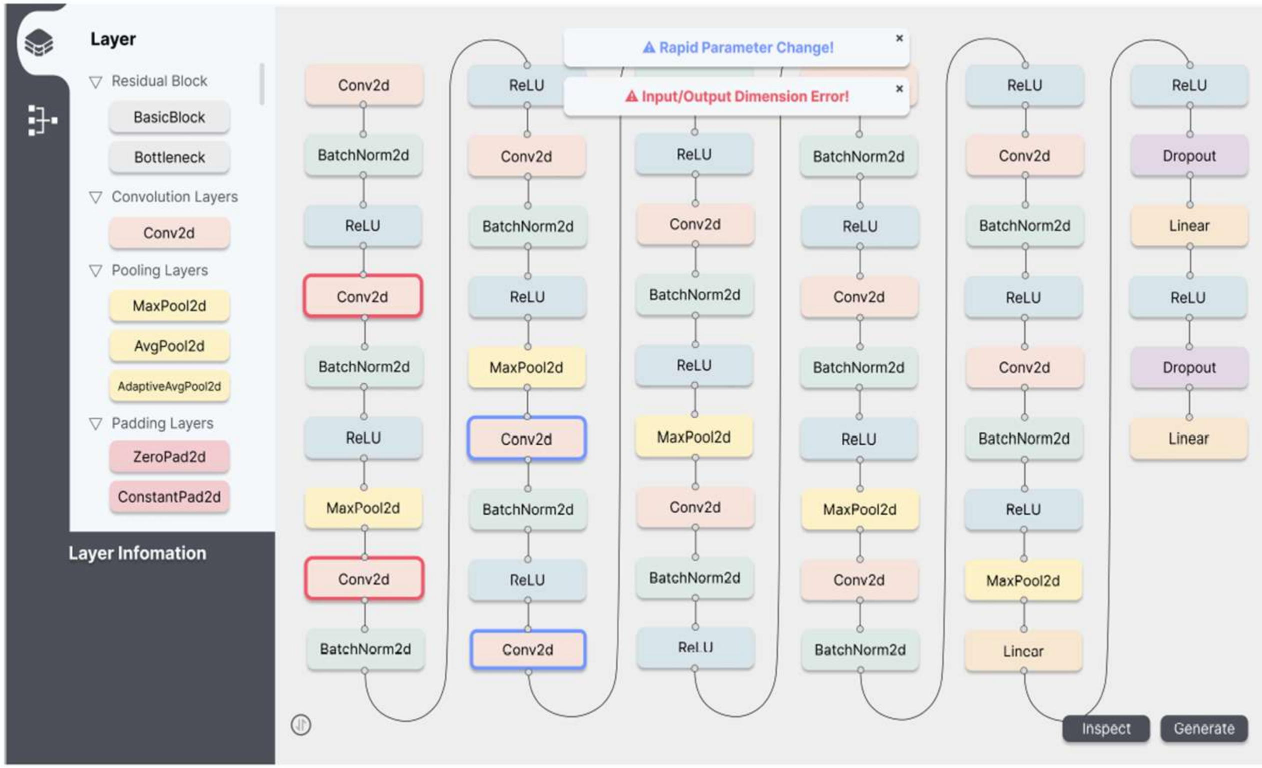
Key Functionalities: Architecture Abstraction



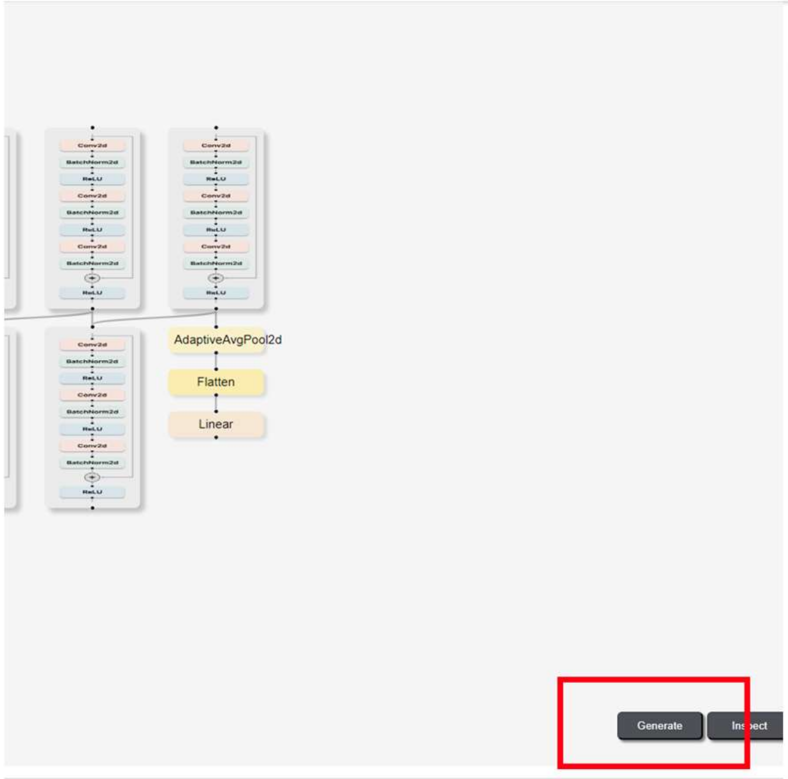
Key Functionalities: Architecture Validation

- **Validation of Dimension Compatibility** checks whether the input/output dimensions of adjacent nodes in the architecture match

- **Validation of Parameter Variability** examines whether the number of parameters between adjacent nodes changes significantly.



Key Functionalities: Architecture-to-code Conversion

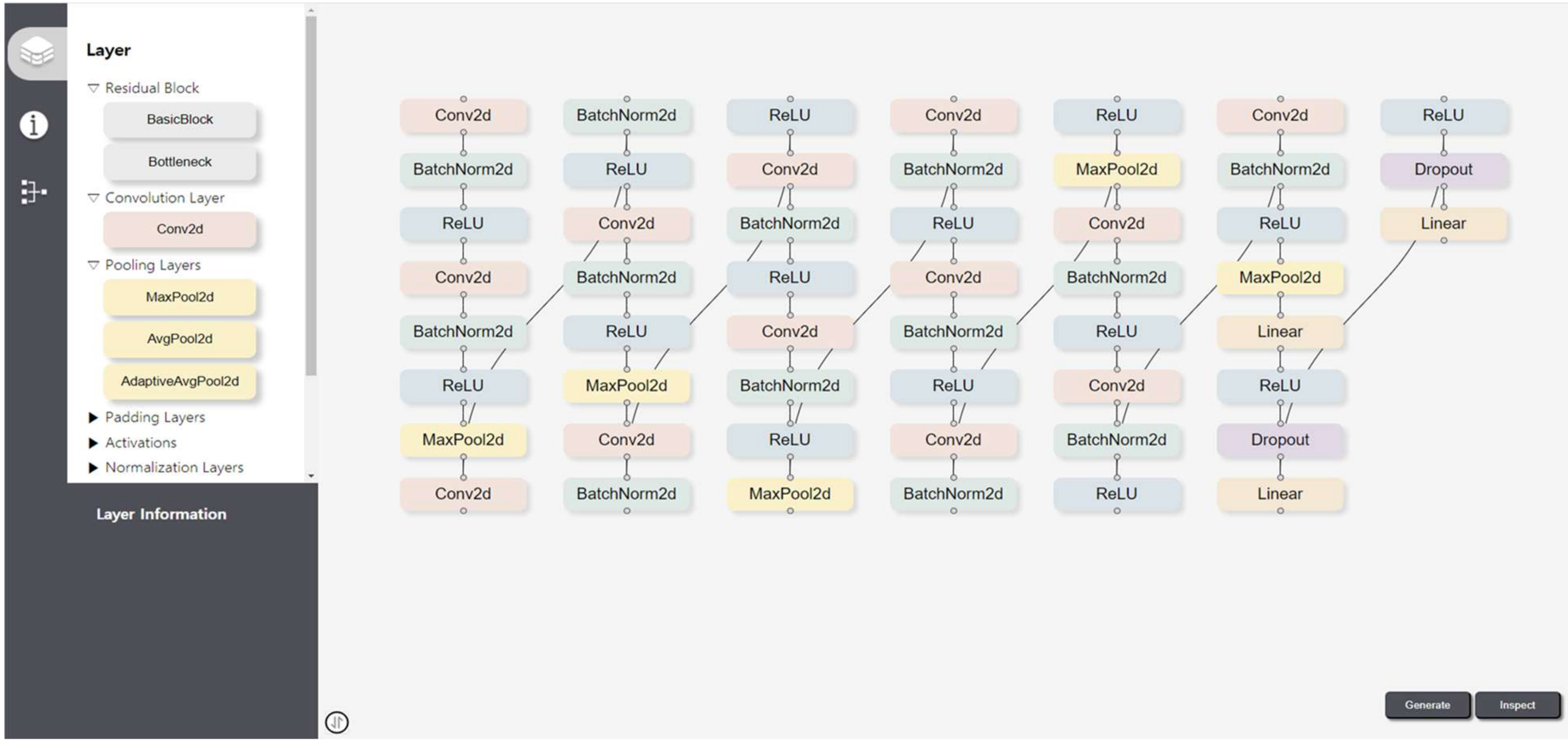



```
import torch.nn.functional as F
import torch.utils.model_zoo as model_zoo
import torch.onnx
```

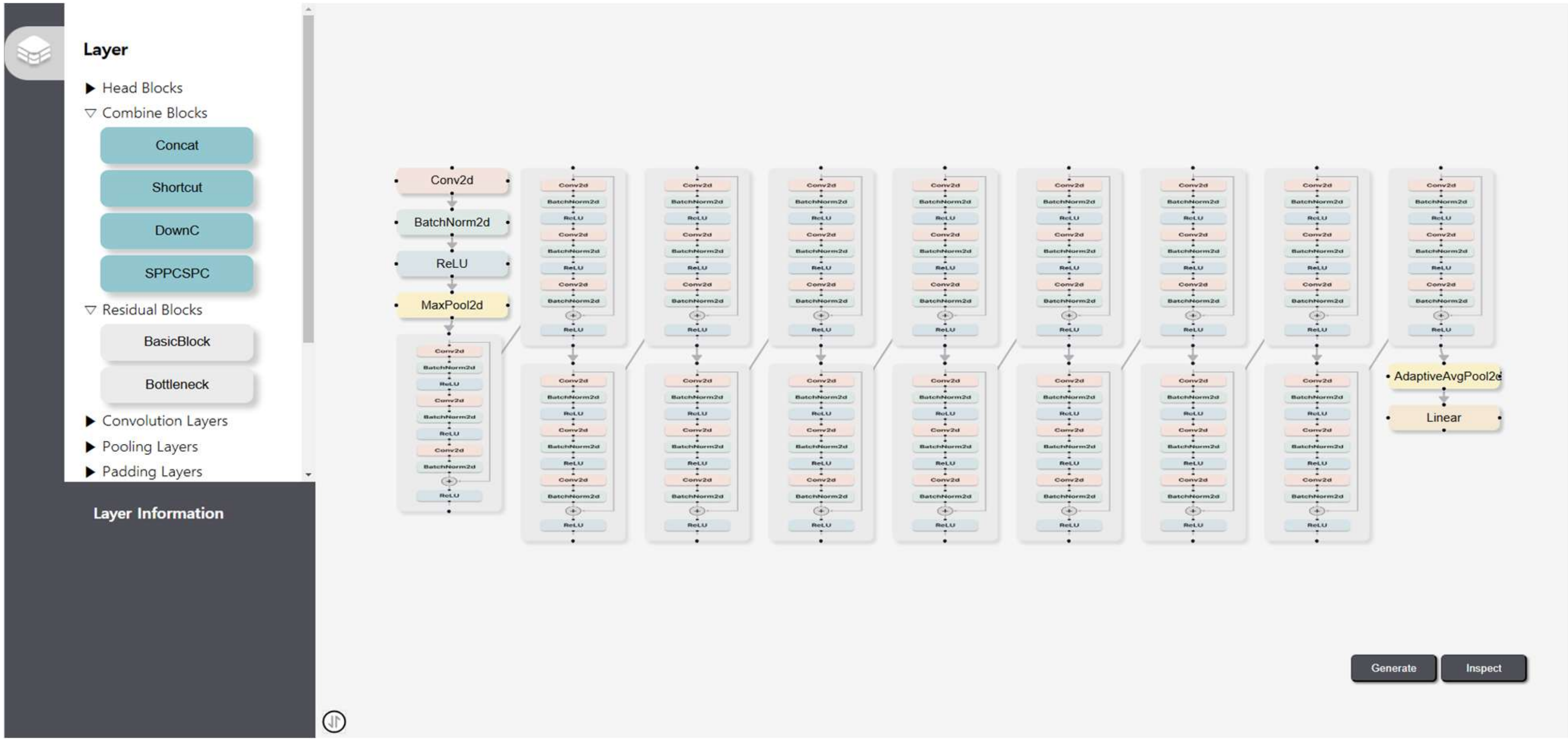
```
path = "/content/drive/MyDrive/model_em4byVKR.pt" # viz로 생성한 pt 파일 load
model = torch.load(path)
print(model)
```

```
Sequential(
  (1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): ReLU()
  (4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): ReLU()
  (7): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0), dilation=1, ceil_mode=False)
  (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): ReLU()
  (11): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (12): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): ReLU()
  (14): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0), dilation=1, ceil_mode=False)
  (15): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (16): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (17): ReLU()
  (18): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (19): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (20): ReLU()
  (21): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (23): ReLU()
  (24): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0), dilation=1, ceil_mode=False)
  (25): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (26): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (27): ReLU()
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (30): ReLU()
  (31): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (32): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (33): ReLU()
  (34): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0), dilation=1, ceil_mode=False)
  (35): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (36): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (37): ReLU()
  (38): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (39): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (40): ReLU()
  (41): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (42): ReLU()
  (43): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0), dilation=1, ceil_mode=False)
  (44): Flatten(start_dim=1, end_dim=-1)
  (45): Linear(in_features=25088, out_features=4096, bias=True)
  (46): ReLU(inplace=True)
  (47): Dropout(p=0.5, inplace=False)
  (48): Linear(in_features=4096, out_features=4096, bias=True)
  (49): ReLU(inplace=True)
  (50): Dropout(p=0.5, inplace=False)
  (51): Linear(in_features=4096, out_features=10, bias=True)
```

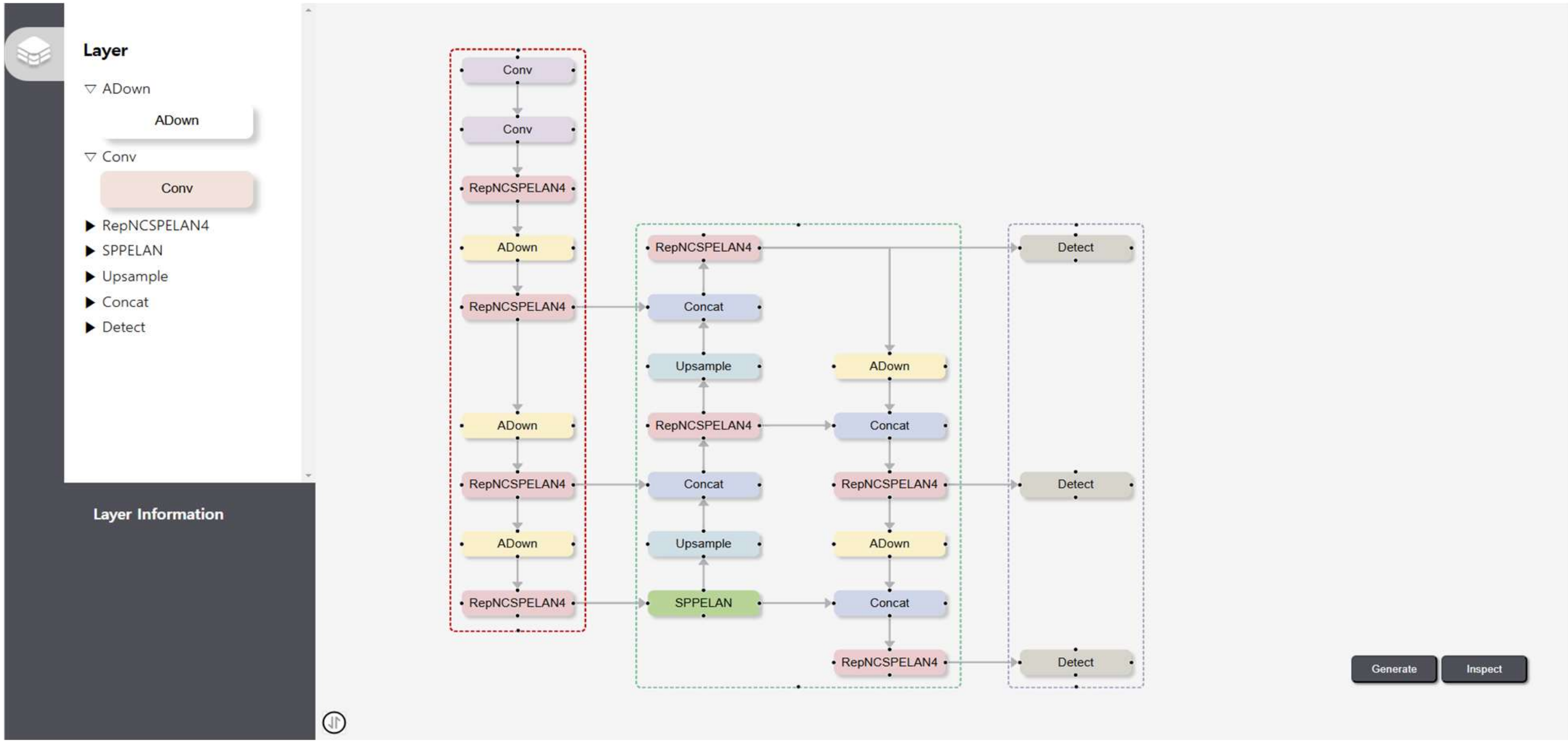
Supported Architectures – VGG16



Supported Architectures – ResNet50

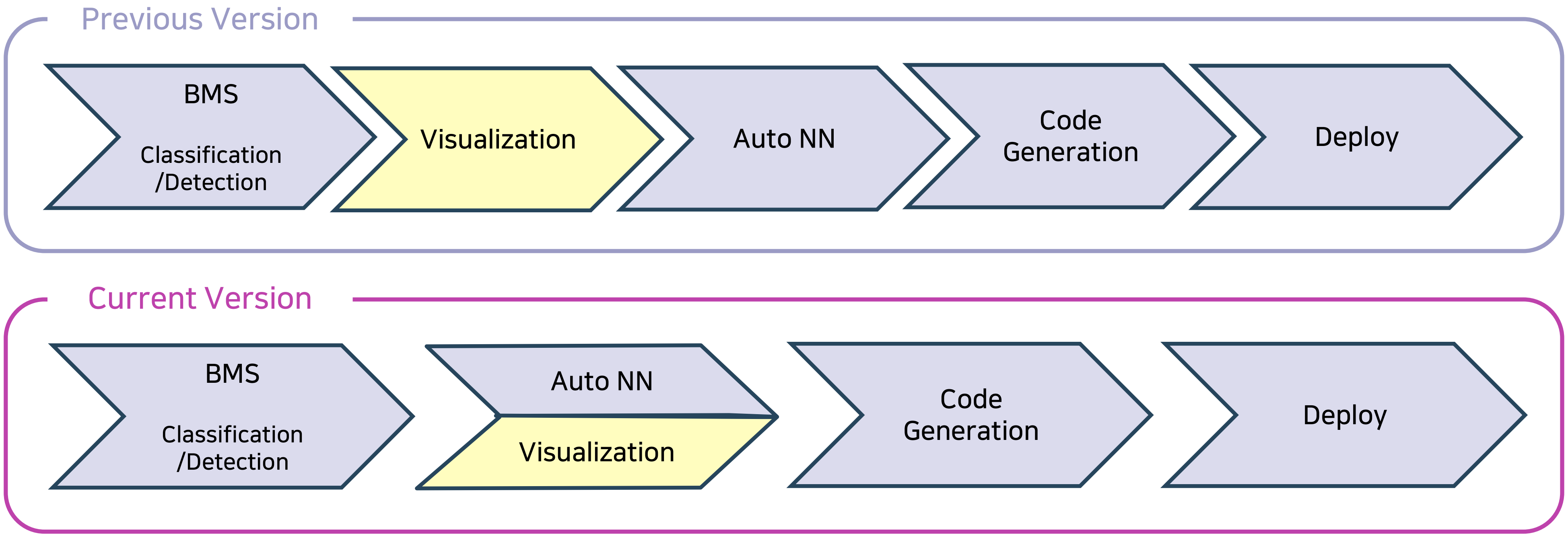


Supported Architectures – YOLOv9



3. VELCRO in TANGO

Workflow



Concluding Remarks

- "No Code!!" NEW visual programming tool for deep learning -

- Non-Expert Modeling

✓ Empowers non-experts to design and develop models in a no-code environment

- Education Support

✓ Reduces the barriers to deep learning modeling

- Prototyping and Validation

✓ Facilitates rapid prototyping and validation of complex architectures

TANGO

감사합니다.

주 관 ETRI (TANGO)

주 최 과학기술정보통신부 IITP 정보통신기획평가원

문 의 parkjb@etri.re.kr / 042-860-5565

후 원 lablup weda tesla System (사)한국인공지능협회 SNUH 서울대학교병원 고려대학교 KOREA UNIVERSITY 홍익대학교 HONGIK UNIVERSITY CAU 중앙대학교 RTst KETI 한국전자기술연구원 AIVN SUREDATA ACRYL 하늘소프트 TTA 한국정보통신기술협회 Telecommunications Technology Association