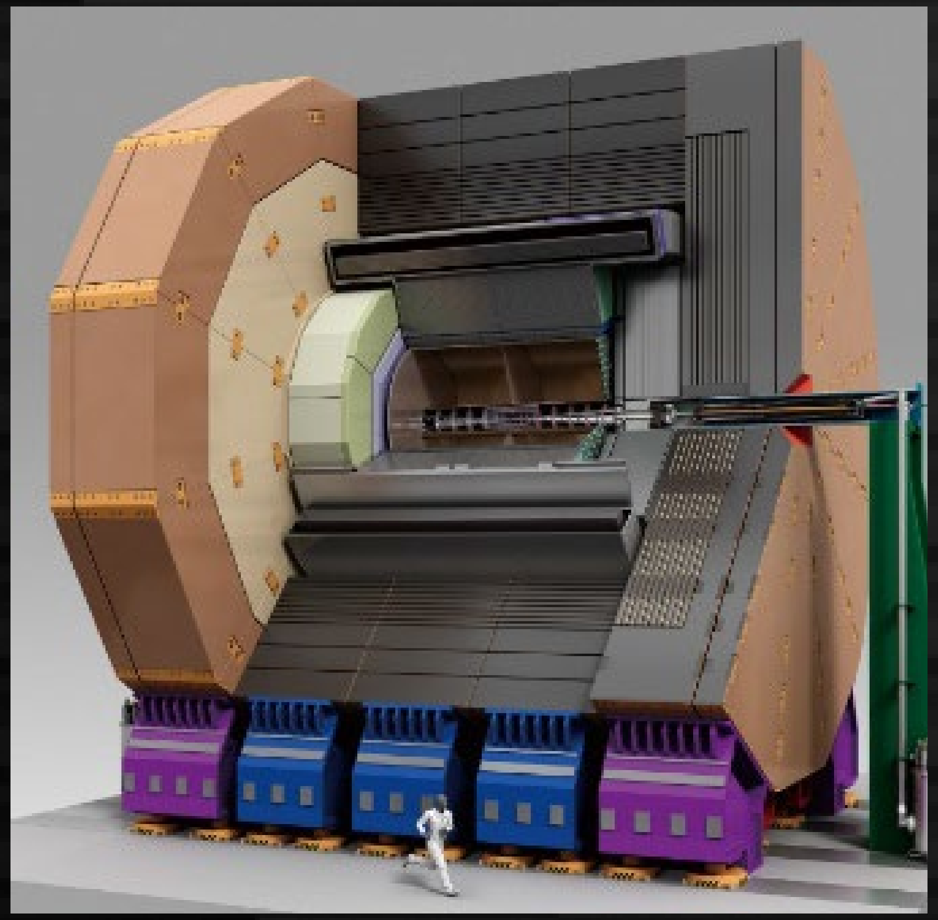
ILCのための深層学習を用いた崩壊点再構成手法の開発

Taikan Suehara
Kyushu University

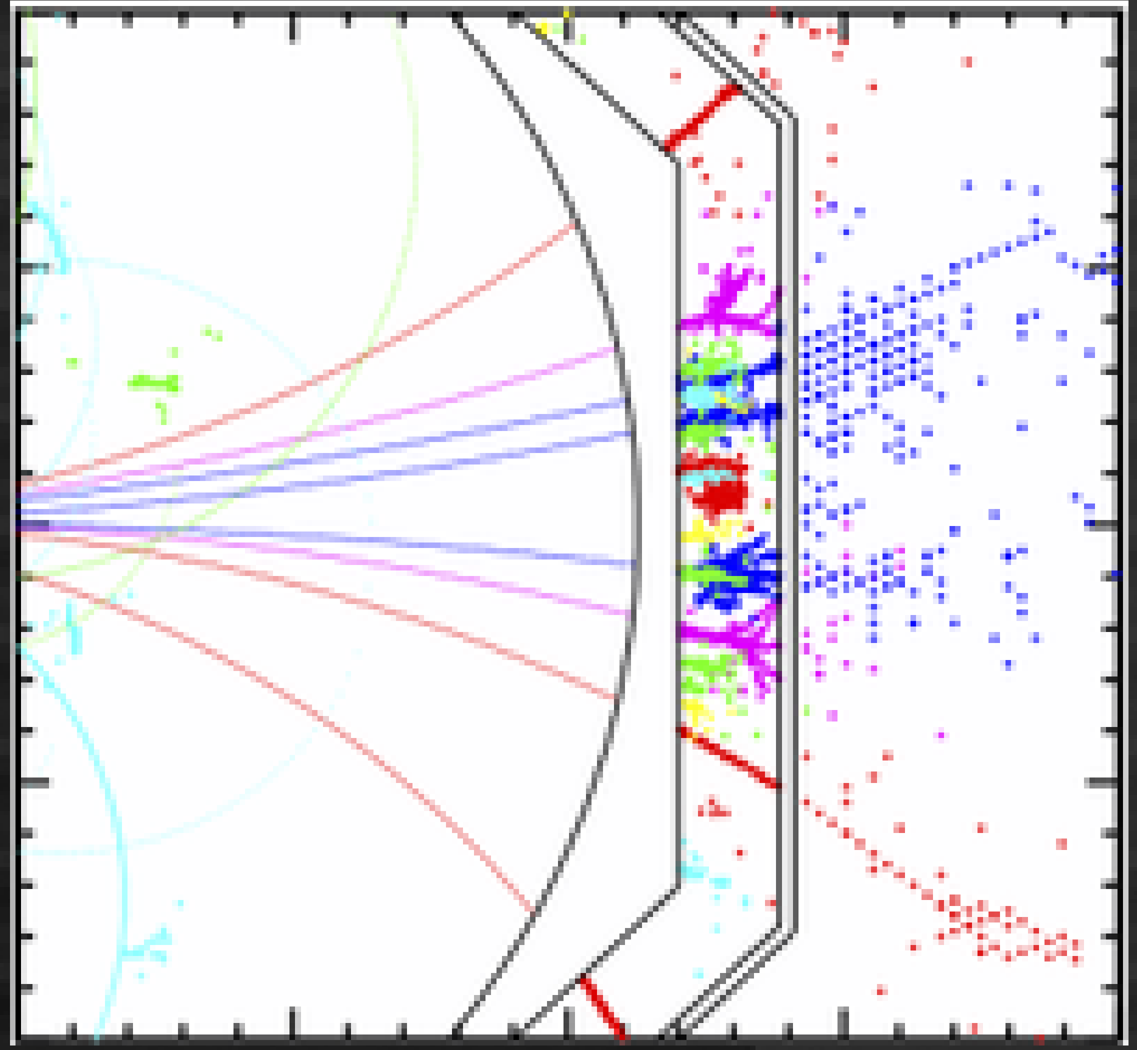
Mainly based on the work done by
K. Goto (for his master thesis, Mar. 2021)

ILCと機械学習

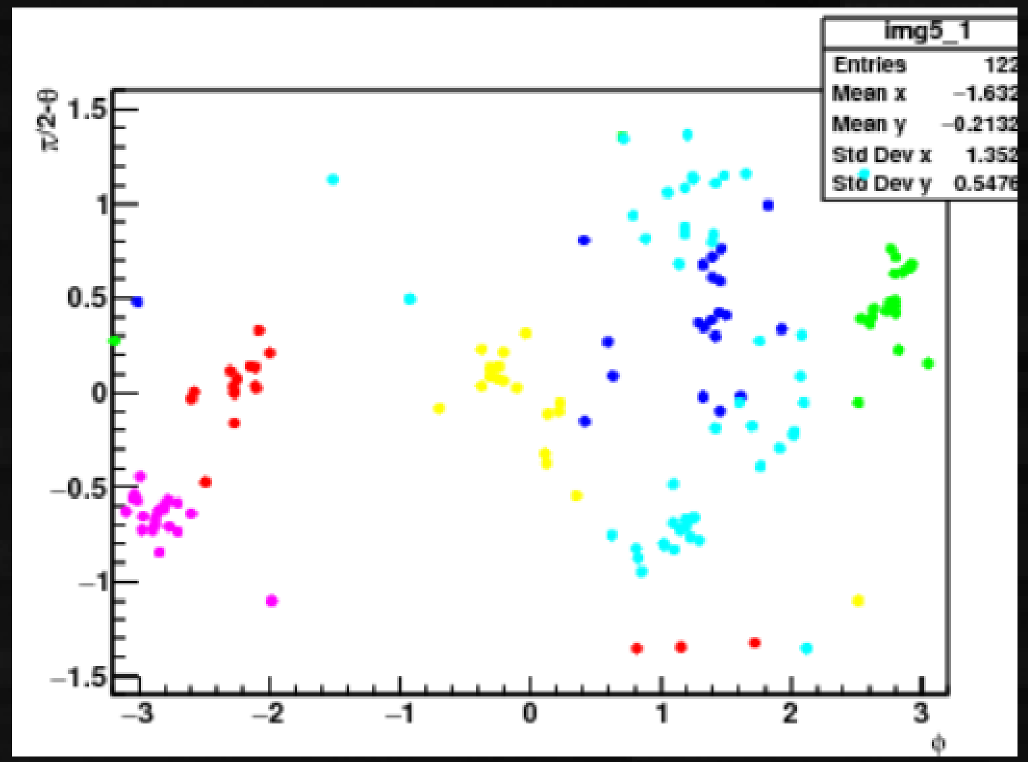
- ILC測定器=ビッグデータ測定器
  - 高精度、高解像度
    - 特にカロリメータの分割度が高い
  - 高度なパターン認識が必要
    - Particle Flow Algorithm
    - ジェット再構成　等
- 深層学習のターゲットが多数
  - ジェット再構成
    - 崩壊点検出、クラスタリング、フレーバー識別
  - PFAの改善
  - 時間情報再構成
  - Simulation, calibration, etc.



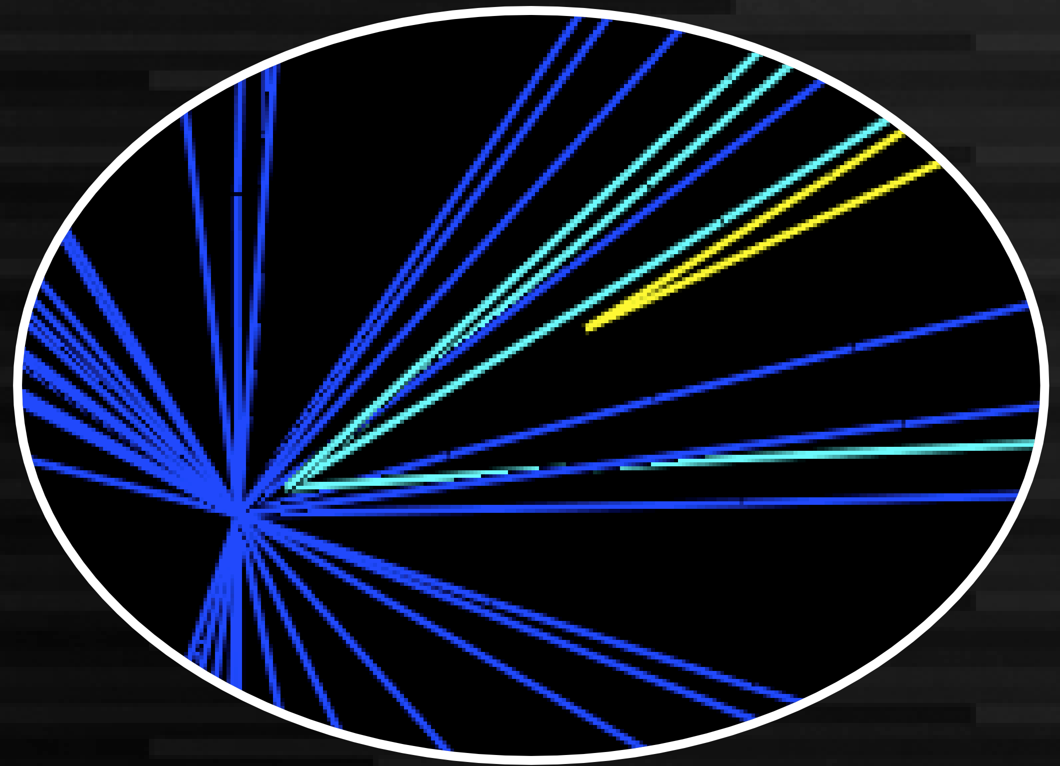
ILD測定器
- Si Vertex (10億ch)
- TPC (連続飛跡検出)
- カロリメータ (1億ch)



カロリメータ内でシャワー発展を直接見ることができる
- PFA (ジェット中の粒子分離)
- 粒子識別 (ToF)



6-jet clustering



2次崩壊点検出･フレーバー識別

Contents

Source codes:
https://github.com/Goto-K/VertexFinderwithDL (python part)
https://github.com/Goto-K/LCFIPlus (adaptation to LCFIPlus)
Papers:
https://arxiv.org/abs/2101.11906
http://epp.phys.kyushu-u.ac.jp/thesis/2021MasterGoto.pdf
(修士論文)

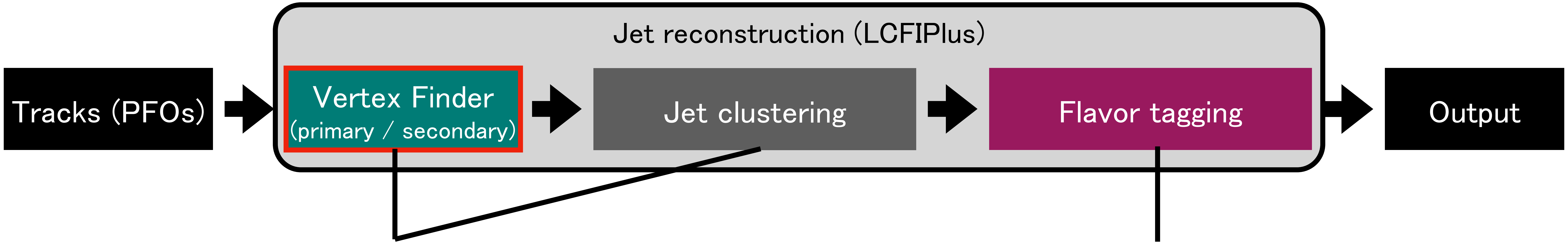
LCFIPlus and flavor tagging

Structure of LCFIPlus

LCFIPlus: Standard flavor tagging software for ILD (also used in SiD, CLICdp, …)

Modular structure to accommodate various algorithms for jet reconstruction

- Vertex finder (primary: tear-down / secondary: build-up)
- Jet clustering (Durham / Valencia-like / K_T) using vertex information; beam-jet rejection incorporated
- Jet vertex refiner (Tuning vertices with jet information; association of vertices to jet when external jet clustering used)
- Flavor tagging (b/c/uds)



Human-tuned process: possibility to be improved
by introducing Machine-Learning techniques

(Traditional) Machine Learning
Boosted Decision Trees (BDT)

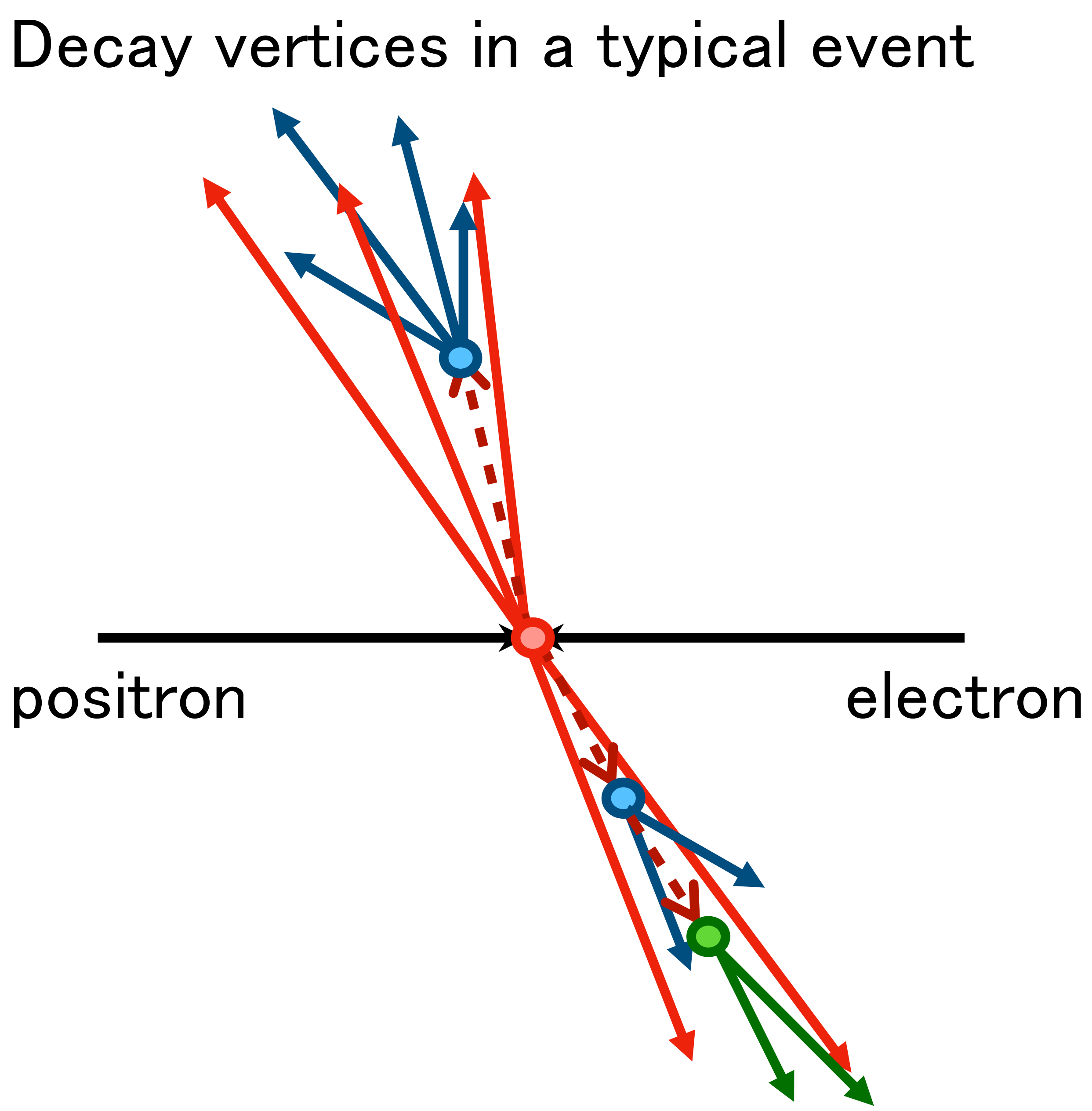
This work: replace Vertex Finder with Deep-Learning (DL) networks as a first step for replacing all jet reconstruction with DL technologies

Vertex finding and simulation conditions

Vertex finding in LCFIPlus

- Build-up method (used for secondary vertices after removing primary/V_0 tracks)

1. Produce a vertex by each track pair (from all tracks, $O(n^2)$ combinations)
2. Select vertices with good quality (cut on χ^2, mass, direction, etc.)
3. Associate additional tracks to the selected vertices (with χ^2 criteria)
4. Associate primary tracks with comparison of χ^2 with primary and selected vertices



For DL-based algorithm,
build-up like method is considered for the network structure

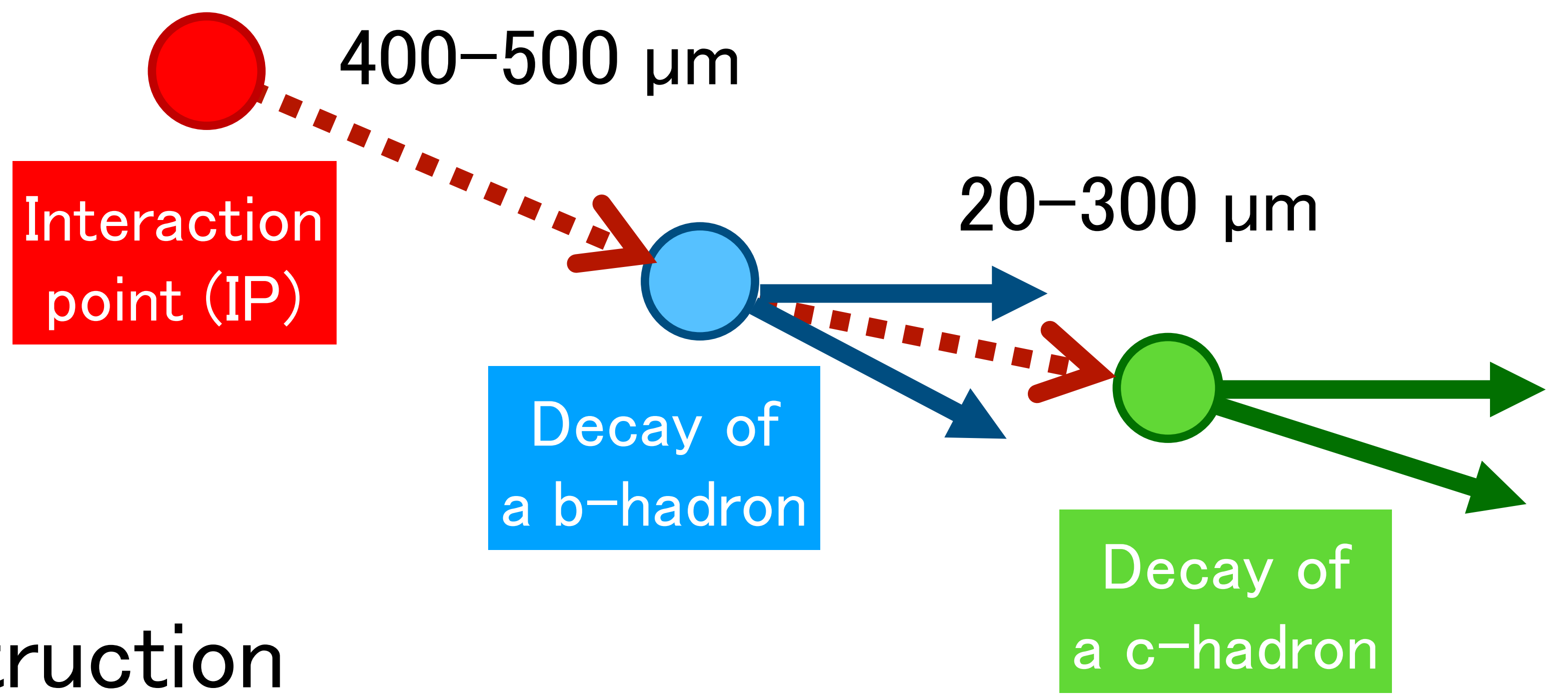
Two neural networks for the DL-based vertex finder

1. Network for selecting track pairs as "vertex seeds"

A Simple feed-forward network currently used

2. Associate tracks to vertex seeds

Recurrent-type neural network is employed

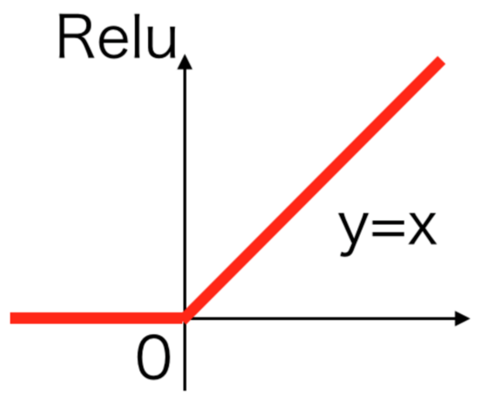


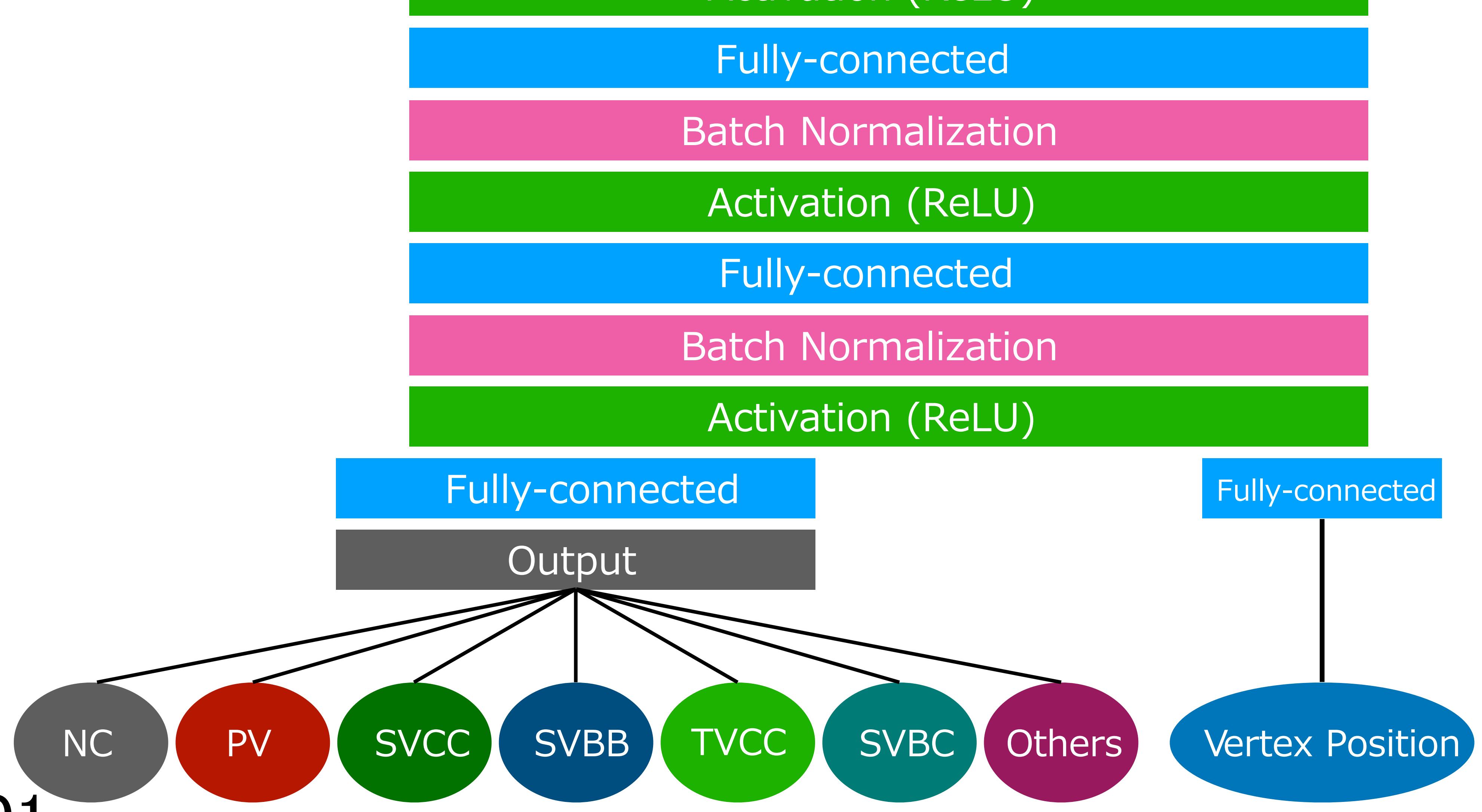
Simulation conditions of this study

- ILD DBD simulation (for comparison with LCFIPlus) / DBD standard reconstruction
- $e^+e^- \rightarrow qq$ (q = b, c, uds) at 91 GeV CM energy, ~500k events each
  (events divided to be used in training and evaluation)

Network design 1: selecting “vertex seeds”

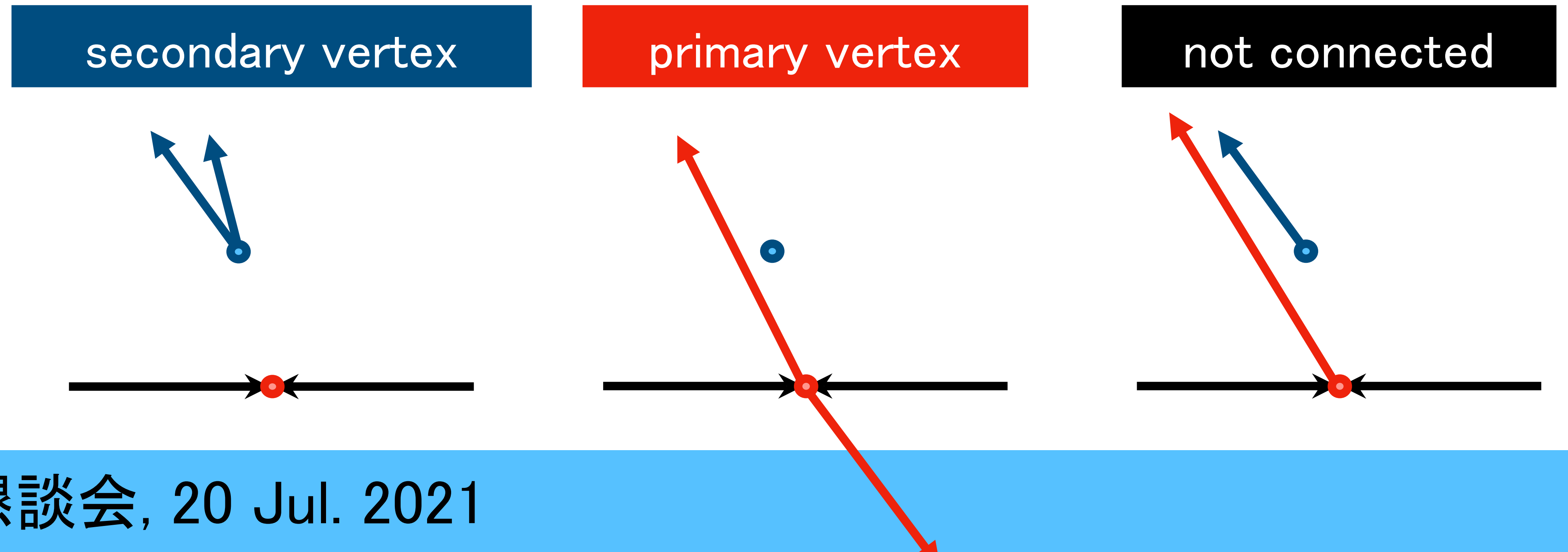
- ➢ Simple feed-forward fully-connected network
- Input: parameters of 2 tracks (total 44 params)
  - Helix parameters (d_0, z_0, ϕ, $\tan\lambda$, Ω)
  - Covariance matrix (15 params)
  - Charge and energy
- 3 fully-connected layer with batch normalization and ReLU activation
- 7 categories for output after final fully-connected layer
  - NC, PV, SVCC, SVBB, TVCC, SVBC, others
- Regression of vertex position with separate fully-connected layer
  - To build “position recognition” algorithm inside the main layers
- Loss function tuned to train categorization and position network
  - 1st step: w(cat, pos) = (0.1, 0.9), 1000 epoch, learning rate = 0.001
  - 2nd step: w(cat, pos) = (0.9, 0.1), 1500 epoch, learning rate = 0.001
  - 3rd step: w(cat, pos) = (0.95, 0.05), 500 epoch, learning rate = 0.0001
- PV and SV/TVxx categories are used for “vertex seeds”



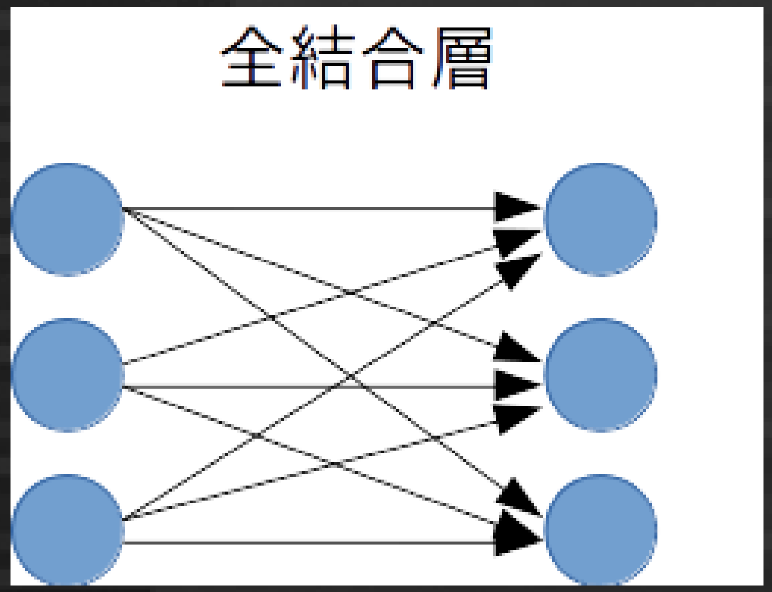


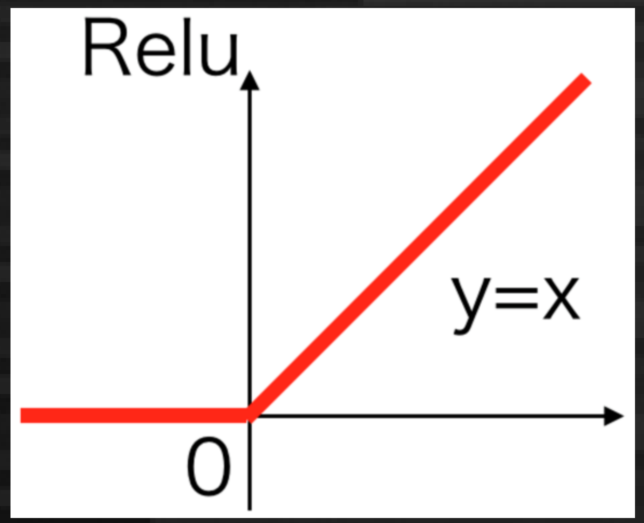
PV: both tracks from primary vertex
NC: track coming from different vertex
SVBB: both tracks from b hadrons in $e^+e^- \to bb$ samples
TVCC: both tracks from c hadrons in $e^+e^- \to bb$ samples
SVBC: one track from b, the other from c in bb samples
TVCC: both track from c hadrons in $e^+e^- \to cc$ samples
Others: tracks coming from V_0 or other vertices

Seed categories



補足説明1 (基礎知識)





- 全結合型深層学習の基本
  - 全結合層: 前段ノードと後段ノードをN:N接続する
  - Batch normalization: 各ノードの入力をbatch (学習単位) ごとに正規化する
    勾配消失・爆発対策 / 過学習抑制などさまざまな効果
  - 活性層: 活性化関数で非線形化する (従来はsigmoidが使われた)
    ReLU: 勾配が収束(消失)しにくく、多段NNに適している
- (狭義の)深層学習の2タイプ
  - Classification(分類問題): 正答ラベルがdiscrete, 各ラベルの確率をoutput
    - outputをsoftmaxで正規化し、loss functionにcross entropyを用いる
  - Regression(回帰問題): 正答ラベルが連続的, 誤差を最小化する
    - Outputは恒等関数(そのまま), loss functionはmean square error等
    - (fitting等に比べ)精度を出すのはあまり得意ではない (指数関数的に収束しにくい)

softmax

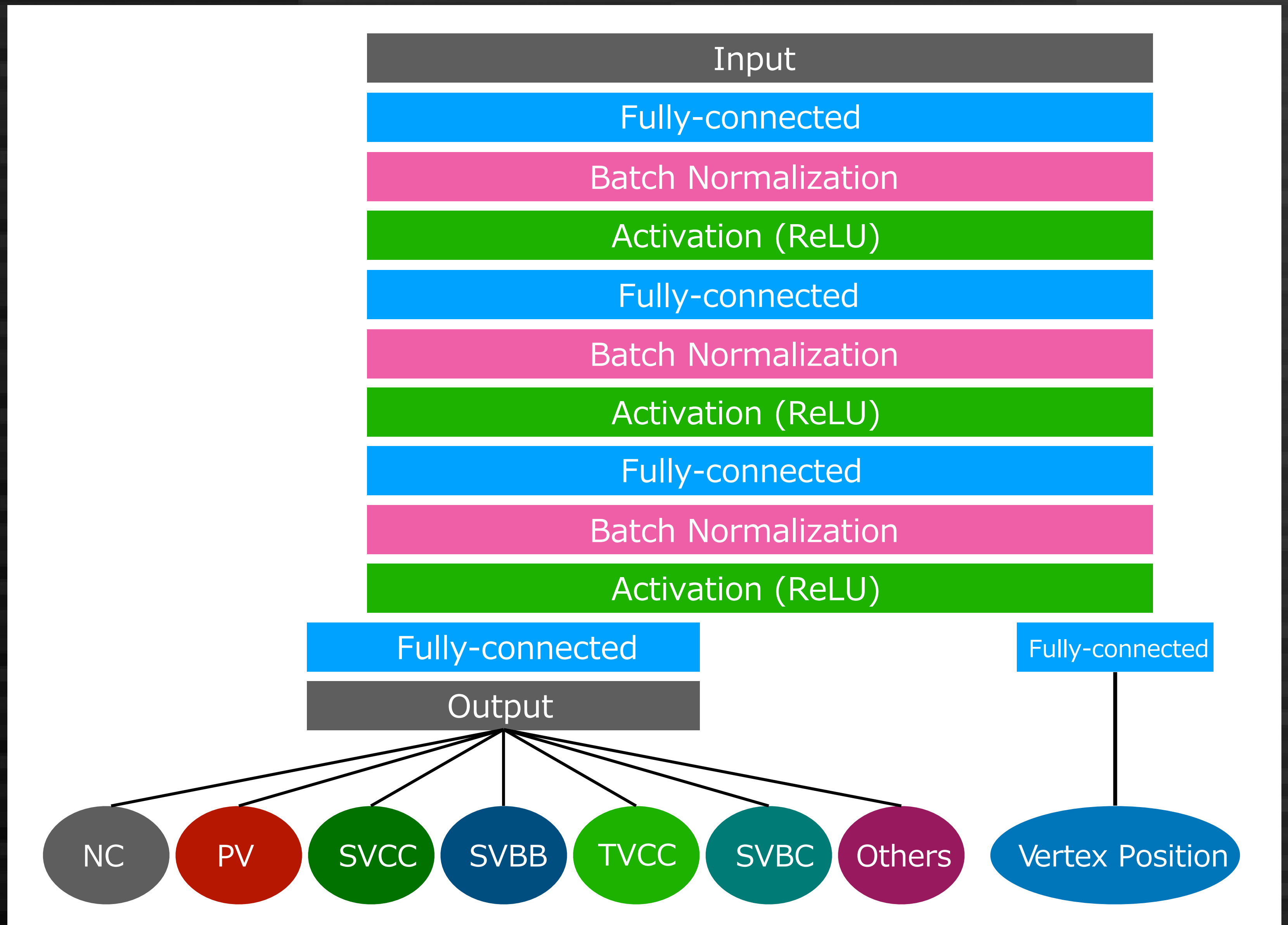
$$f_i(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

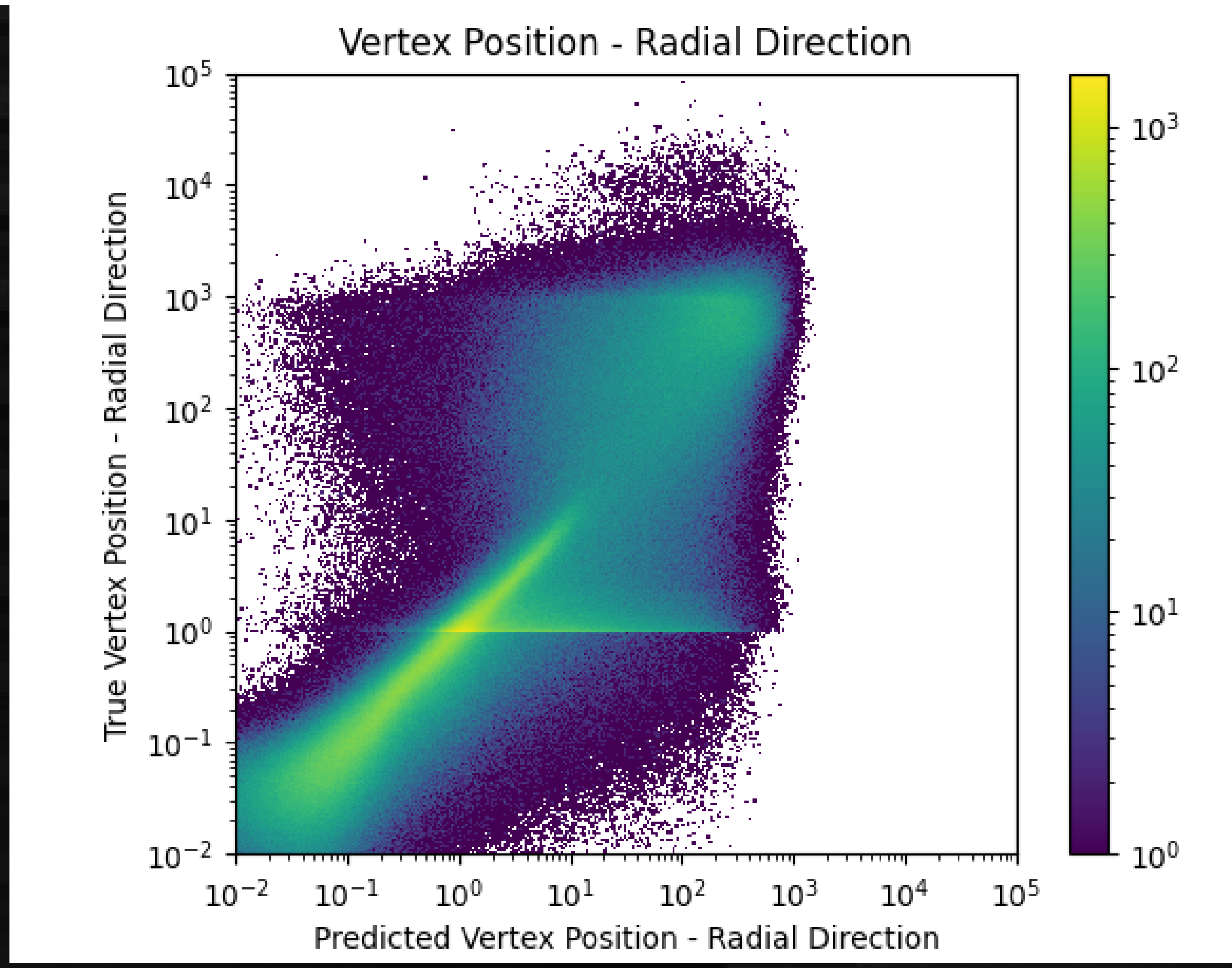
$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

Cross entropy
y_iが正答ラベル (0か1), hatつきが推論のprobability

補足説明2

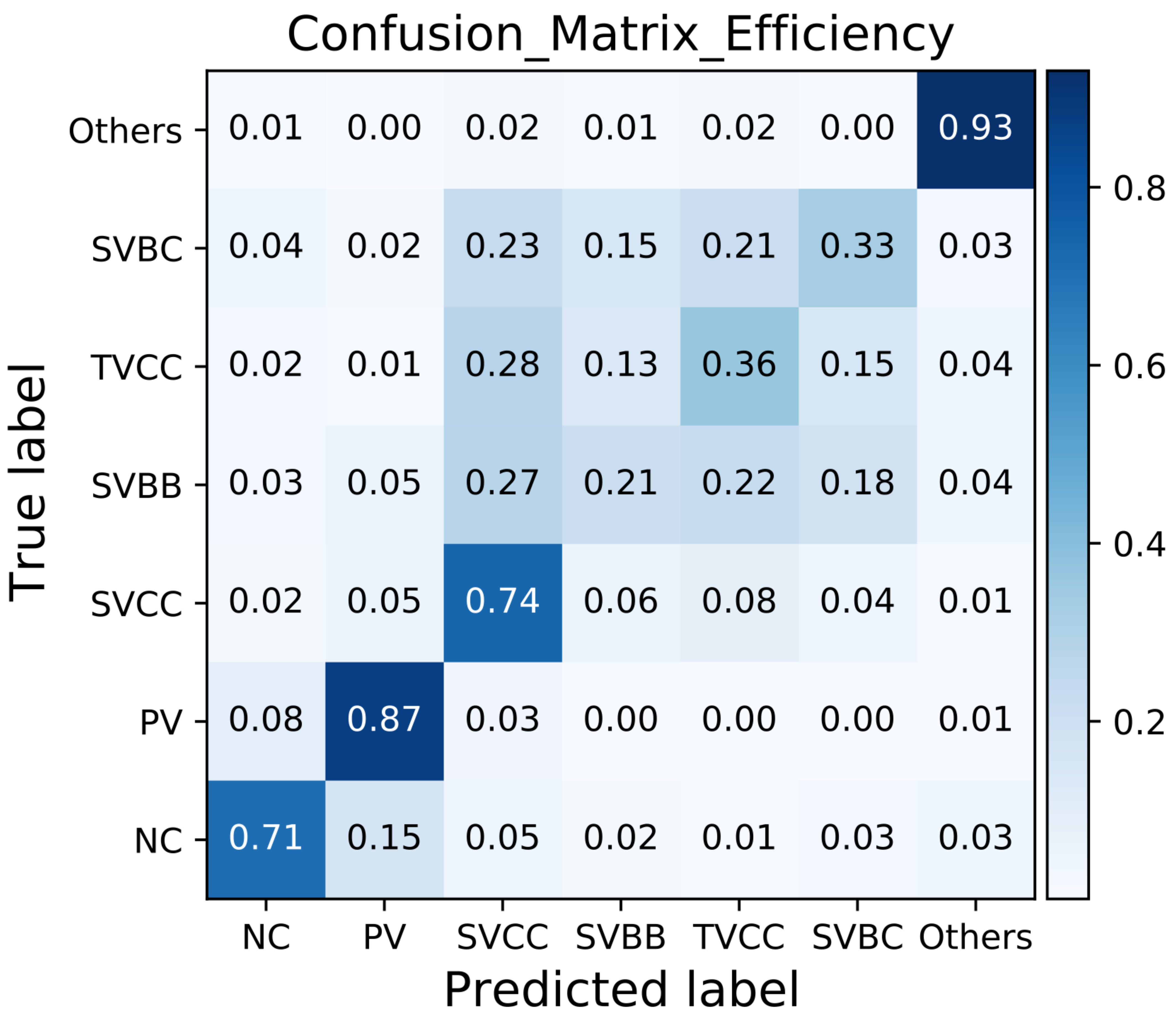
- Input: track parameter x 2
  - Error matrixも含む (使えているか?)
- Vertexの条件
  - 交わっている (MLには困難)
  - Primaryかsecondary(bb, cc, bc etc)か
    - IP付近ならprimary (MLには簡単)
    - あとは距離やmassで適宜判断したい
- Vertex positionを正解ラベルとする部分を追加することでpositionを求めるネットワークを分類問題にも使ってほしい
  - Loss functionはtrainingの初期はpositionを主に、後半はcategoryを主に学習させる

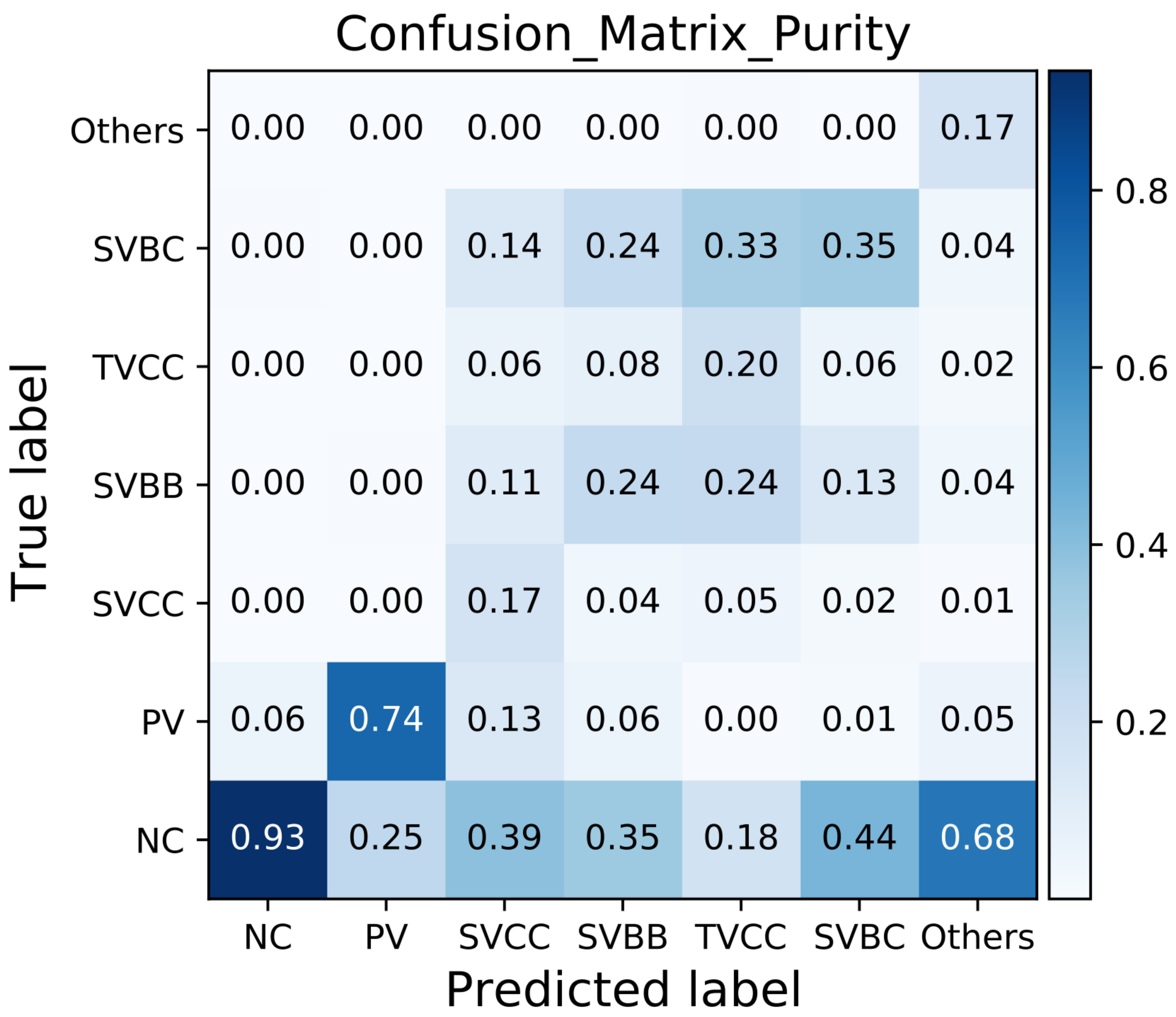




2. 崩壊点検出の為のニューラルネットワーク

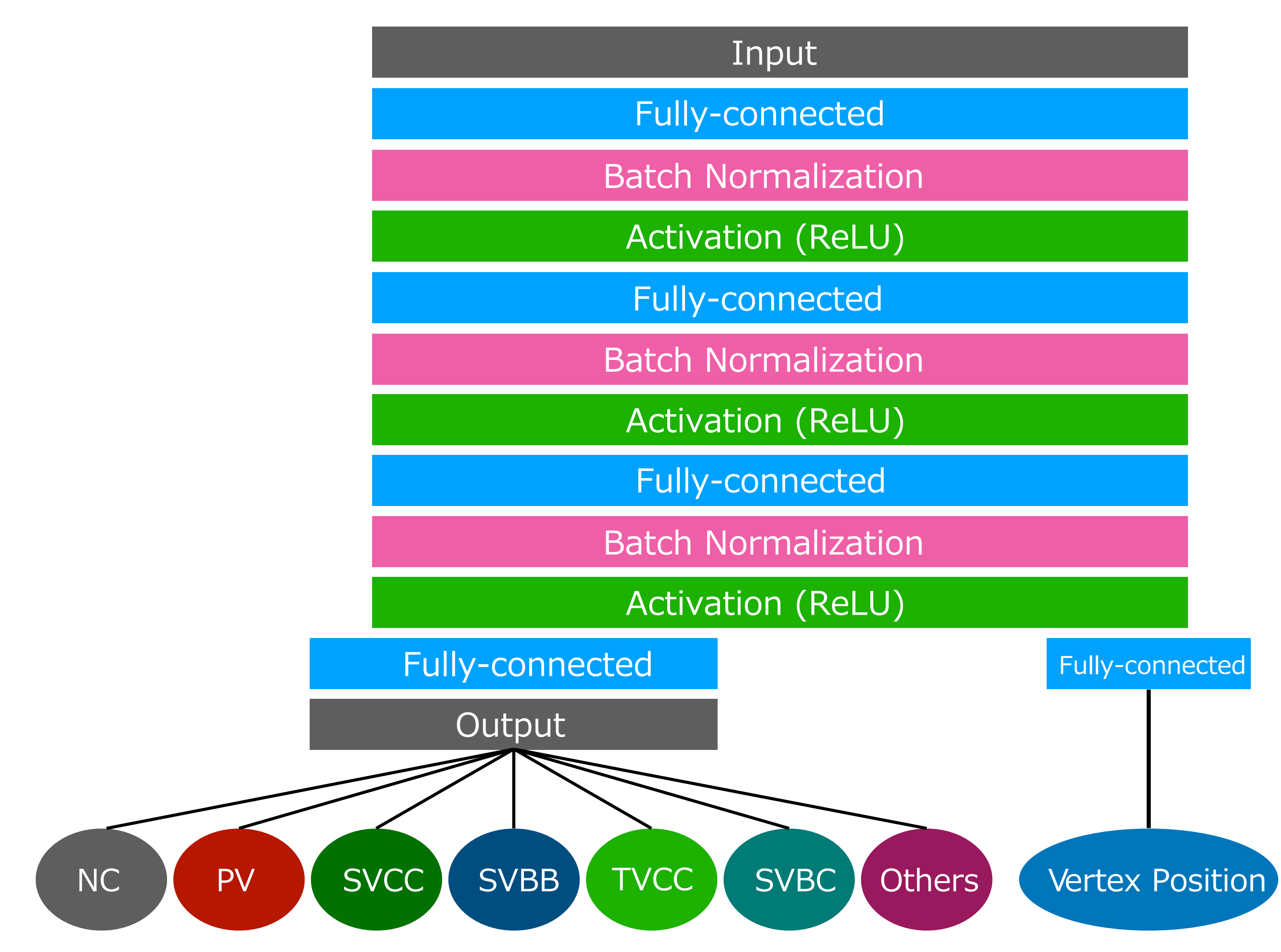
飛跡対についてのネットワーク –構造と性能–

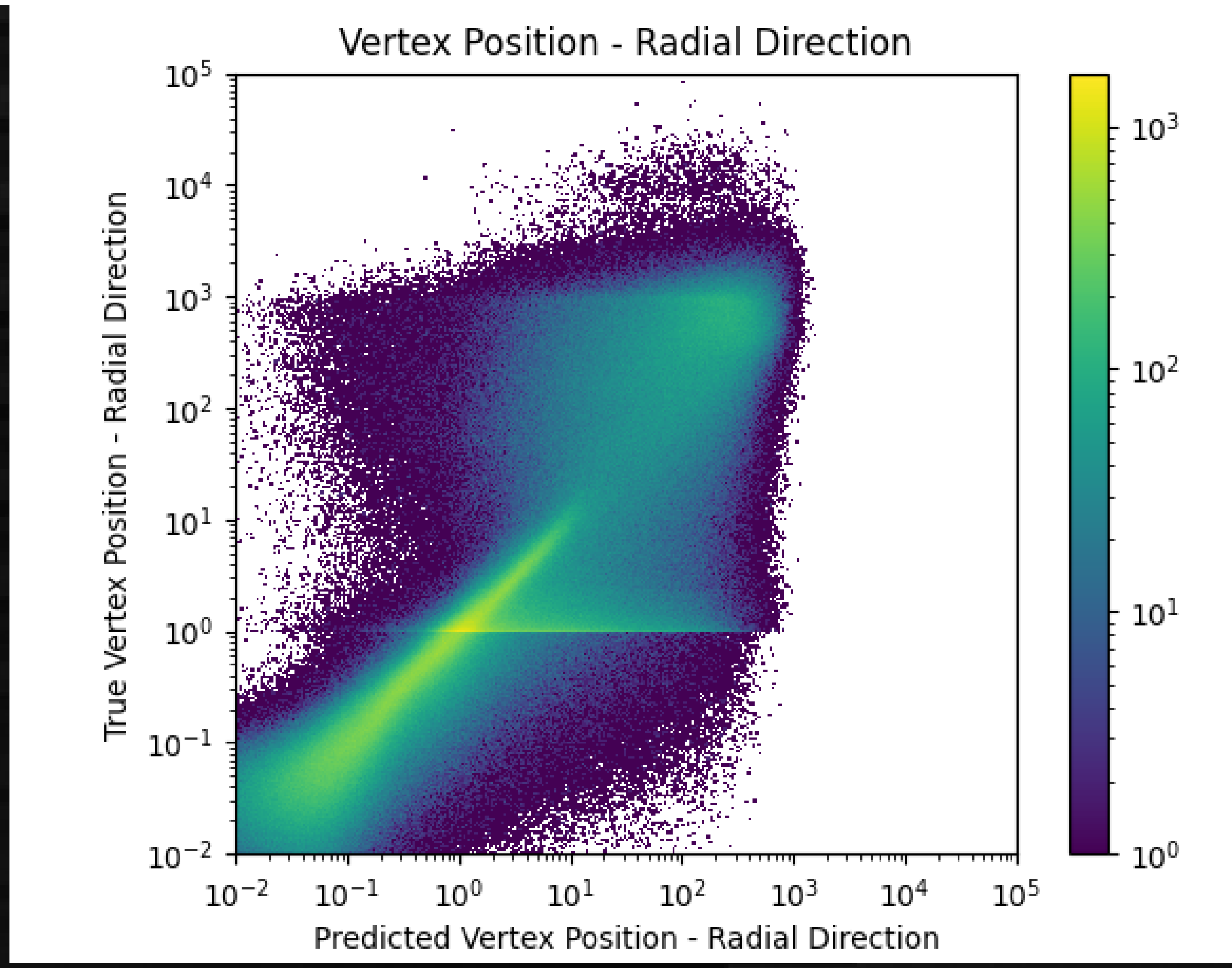




補足説明3 (今後のプラン)

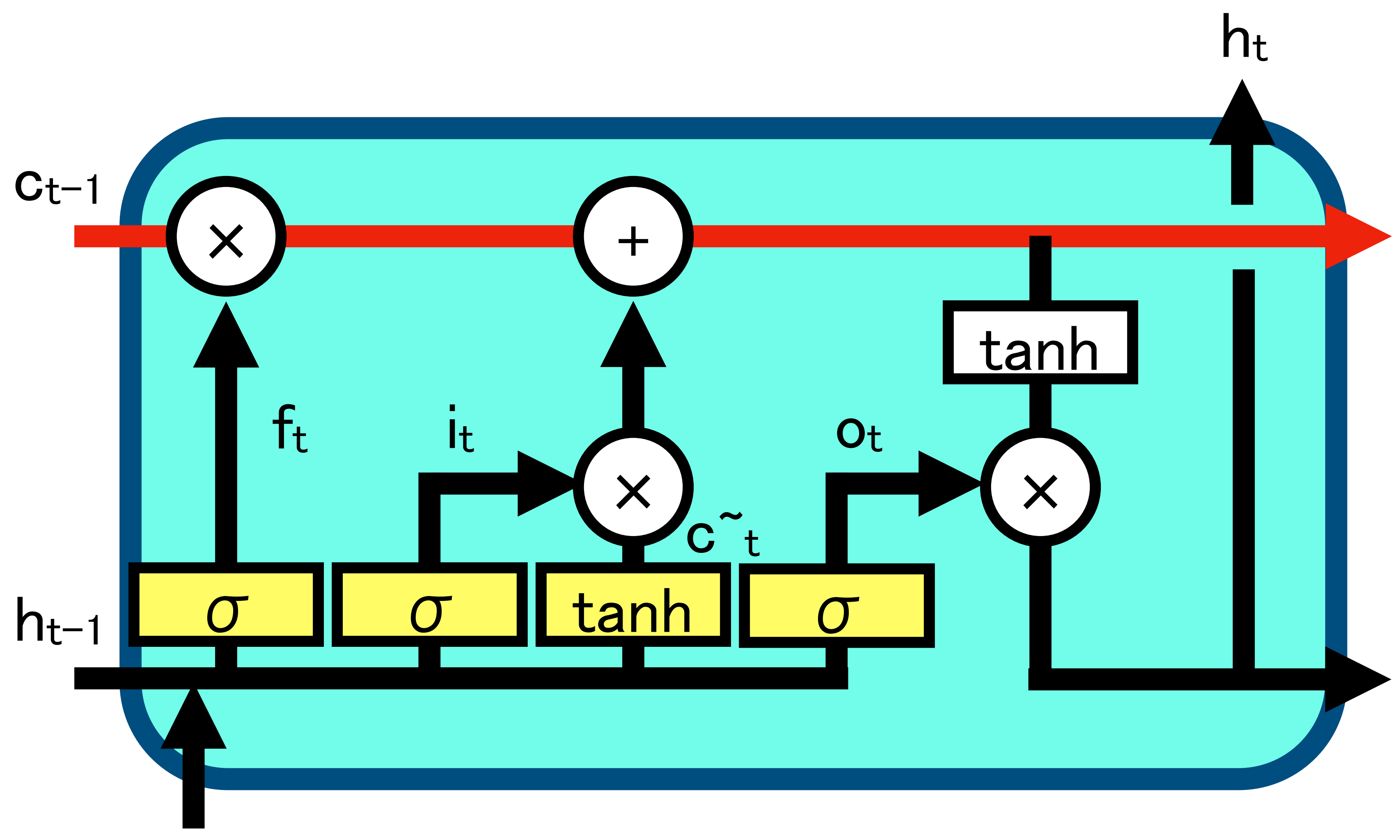
- 交点を求めるアルゴリズムをまじめに考える
  - トラックのパラメータ表示
    (x, y, zをtで表示)
    → x1-x2, y1-y2, z1-z2をtで2次元行列に
    → 交点付近を取り出してNNにかける 等
  - Error matrixをどう取り込むか
- Fitterの結果をあらかじめ与える
  - 将来の高速化の観点からあまりやりたくない
  - 性能の比較には有用
- 別の方法
  - Topological vertex finder的な
    - Network(orそのoutput)を「データの抽象表現」ととらえる
  - 交点を求めるネットワークは何かの形で活かせるか





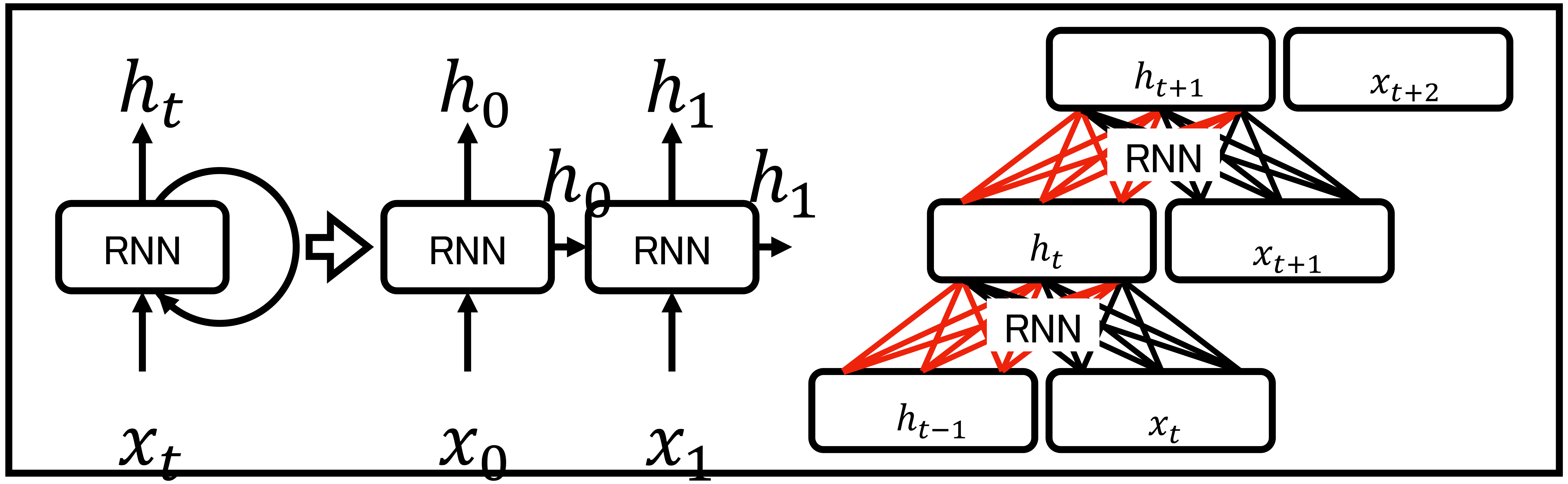
Recurrent Neural Network (RNN) and variance

➢ Recurrent Neural Network
- Neural network designed for variable input length
- Main application: natural language processing (translation etc.)
- "RNN cell" defines a unit of network structure (with learning weights)
- Each input (x_t) is processed sequentially with the same RNN cell (and same weights)
- "Hidden states h_t" are also inputs of the next cell
- Problem on "gradient loss / gradient explosion" → various RNN cell structures are proposed



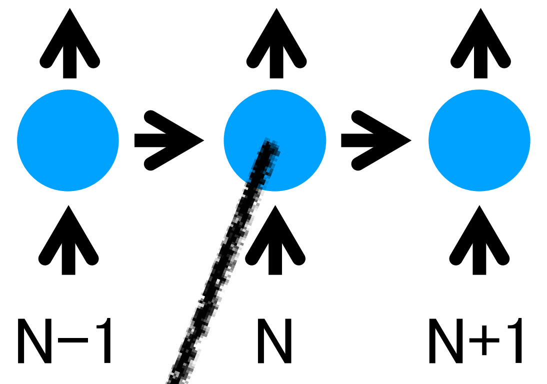
LSTM cell

➢ LSTM (long short term memory)
- One of the RNN cell structure practically used
- "Gate" structures to avoid gradient loss/explosion
  - forget gate
  - input/output gate
- Short-term memory to retain relations to neighbor inputs



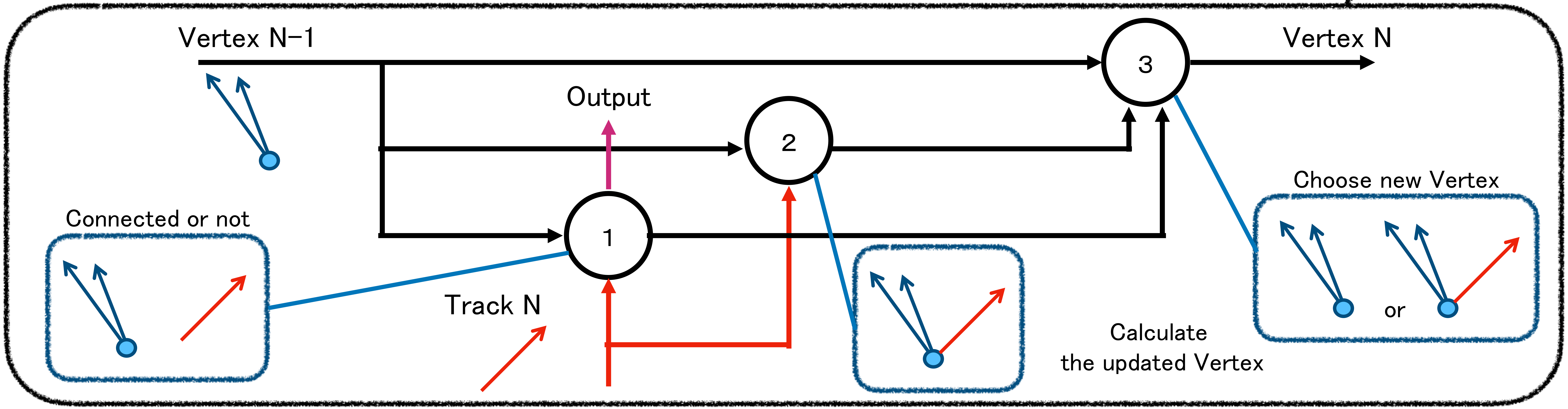
Network design 2: associate tracks to vertices

Custom RNN structure modified from plain LSTM



- No importance in the order of tracks in the track association → "short-term memory" is not necessary
- The custom cell only propagates "long-term memory" which is recognized as "vertex information" to the next cell
- "forget, input and output gates" structures are kept to avoid gradient loss / explosion
- Procedures:

1. Evaluate if the n-th track should be associated or not: $h_N = \sigma(D_h[\sigma(W_o t_N + R_o V_{N-1}) \cdot \tanh(V_{N-1})])$
2. Combine the n-th track and the n-1 th vertex to produce a new vertex:
$$U_N = \sigma(W_i t_N + R_i V_{N-1}) \cdot \tanh(W_z t_N + R_z V_{N-1}) + \sigma(W_f t_N + R_f V_{N-1}) \cdot V_{N-1}$$
3. Combine the old and new vertex according to the evaluation in the 1st step: $V_N = (1 - h_N) V_{N-1} + h_N U_N$



Network design 2: additional features

Attention mechanism

- State-of-the-art scheme of machine learning to specify “attention” to certain elements of the network
- Usually used in encoder-decoder models

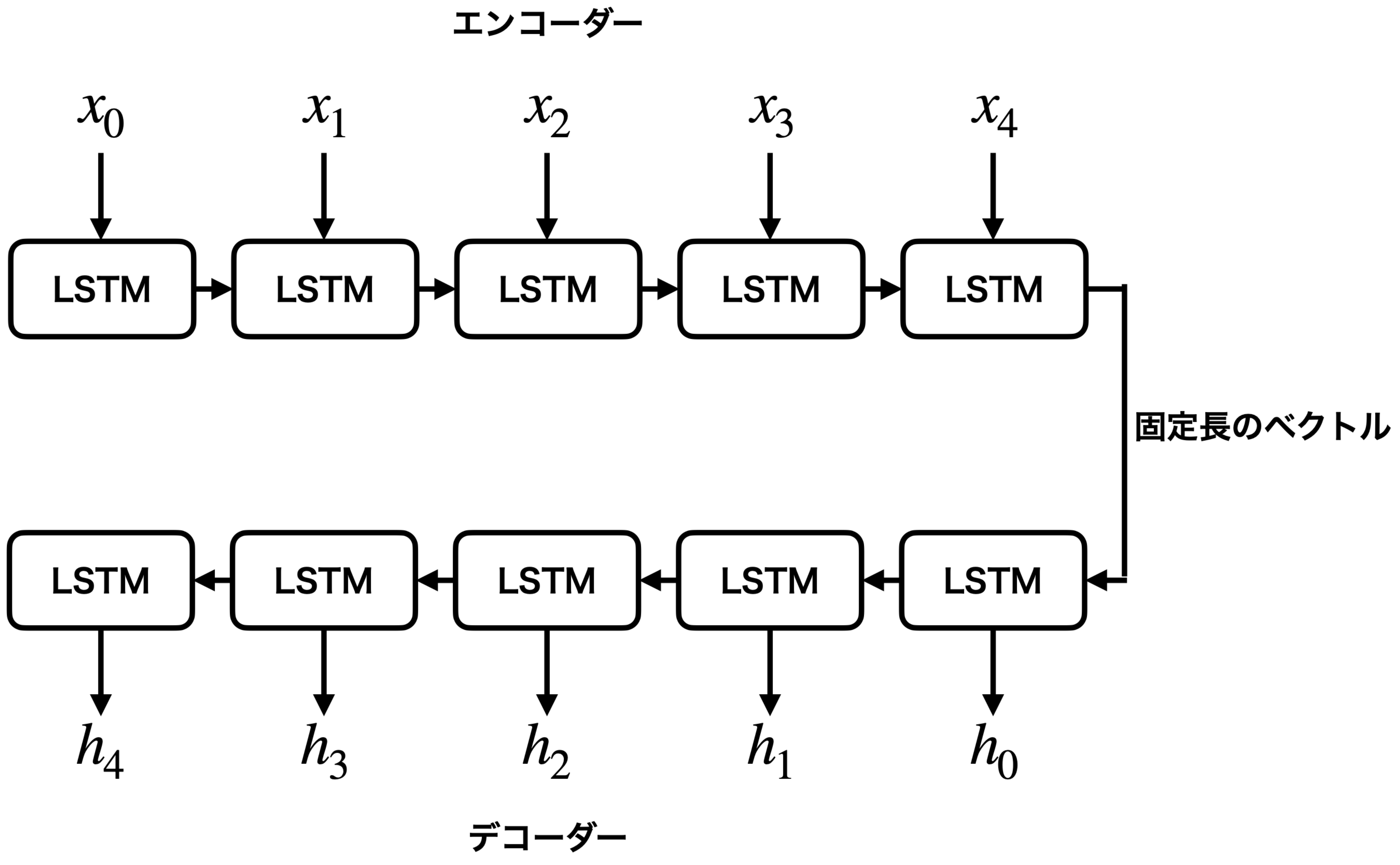
Encoder-decoder model

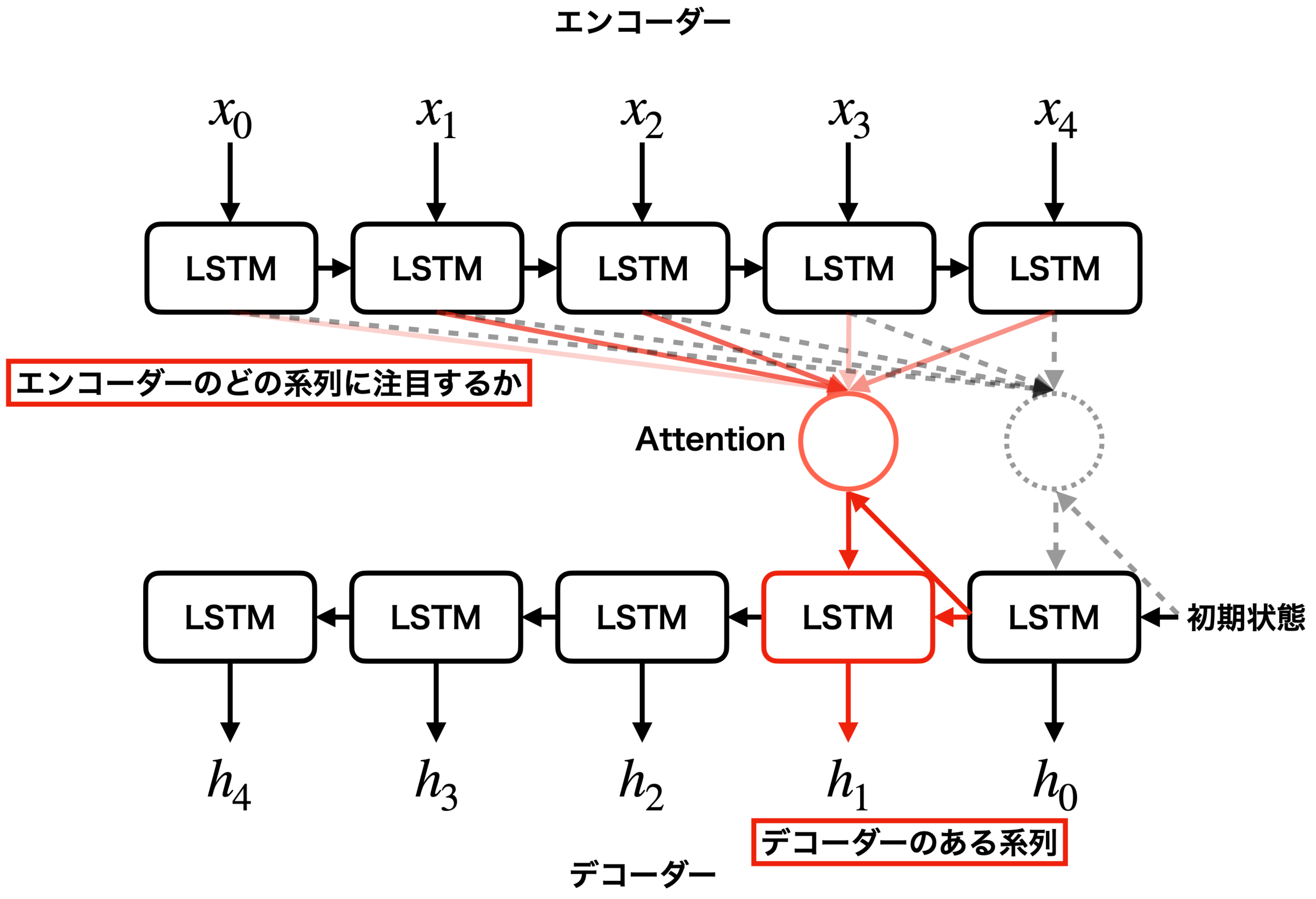
- Encoder: making array of “hidden states” forming storage of information (“vertex” in our case) at the output
- Decoder: derive the essential information from the encoder output (in our case “associated” or not)

Attentionの説明

Attention

- 情報のある部分に注目させるための技術
  - 代名詞が何を意味しているか
  - 質問に対する答えの位置 など…
- 単純なエンコーダー・デコーダーでは系列の長さに関わらず、同じ大きさの情報で伝達してしまう
- Attentionはエンコーダーに応じて、情報を確保できる





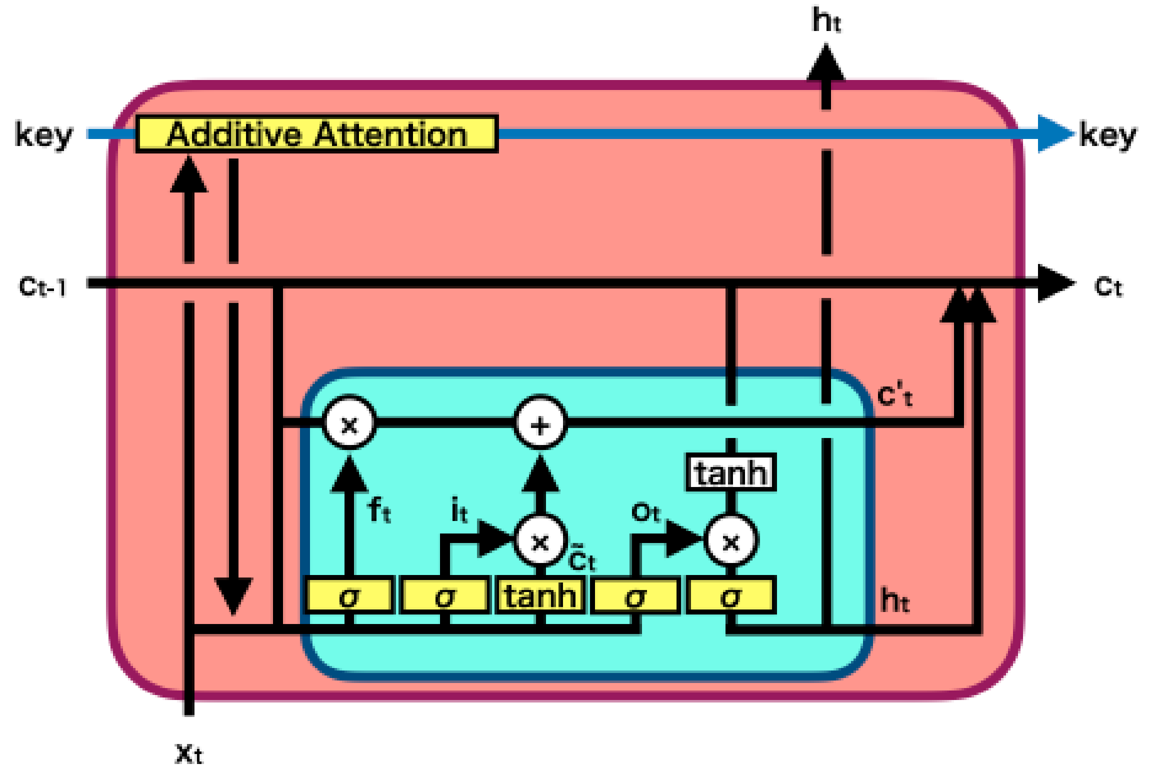
Attentionの説明

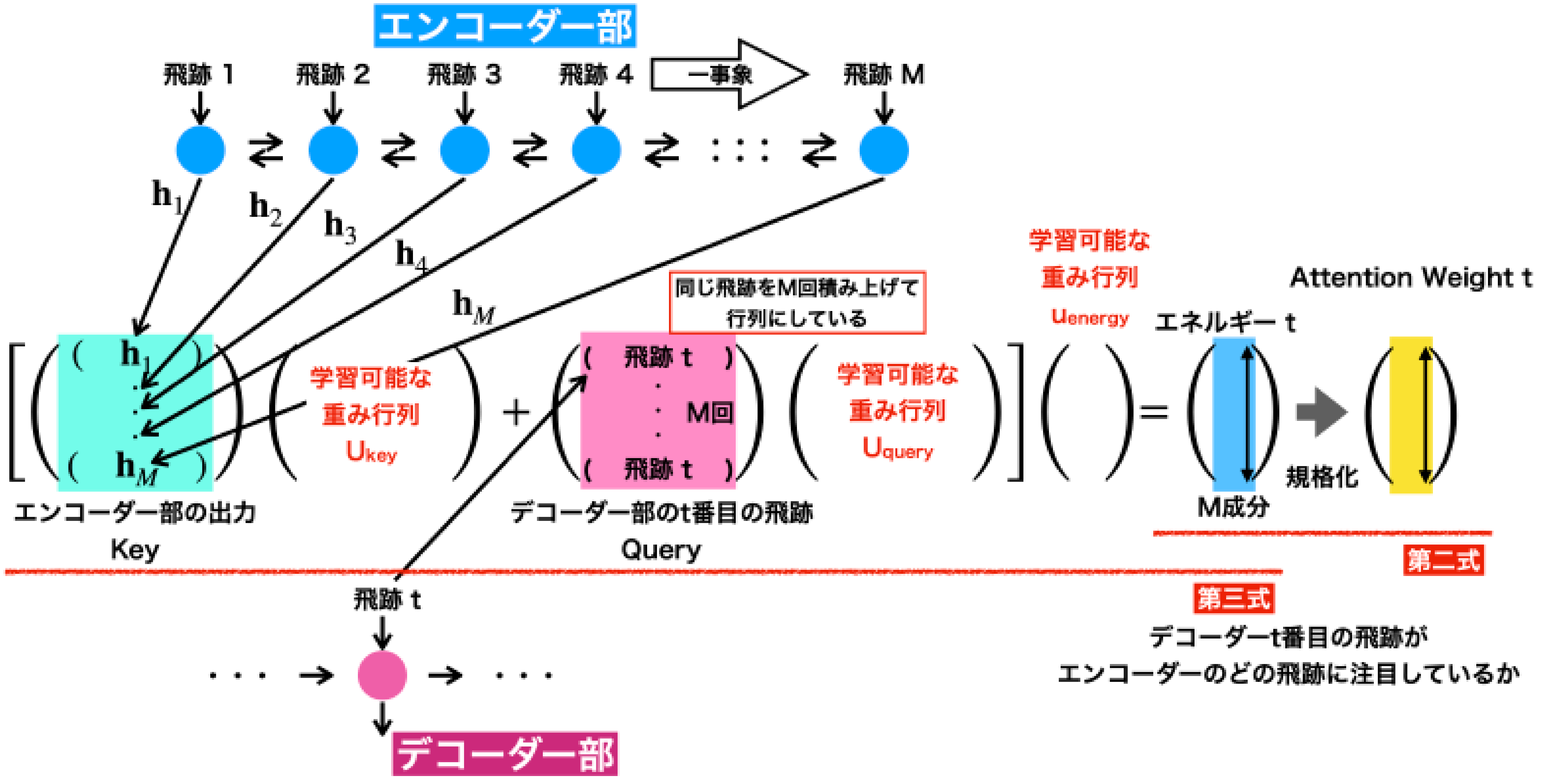
Additive attention: arXiv: 1409.0473

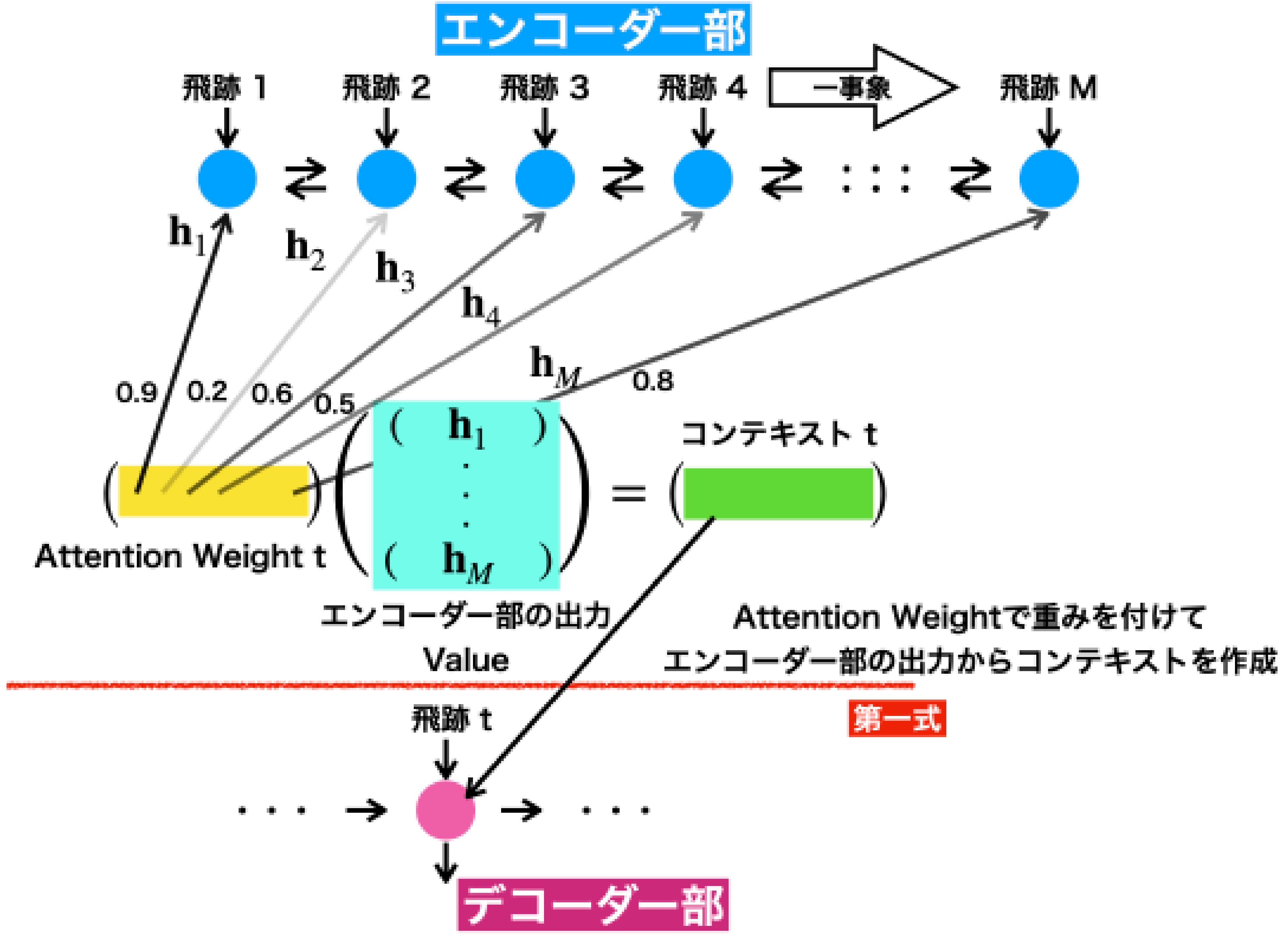
$$\gamma_t = \alpha_t V$$

$$\begin{aligned}\alpha_t &= (\alpha_{t,0},\ \alpha_{t,1},\ \alpha_{t,2},\ \cdots \alpha_{t,i},\ \cdots) \\ &= \left(\frac{\exp(e_{t,0})}{\sum_j \exp(e_{t,j})}, \frac{\exp(e_{t,1})}{\sum_j \exp(e_{t,j})}, \frac{\exp(e_{t,2})}{\sum_j \exp(e_{t,j})}, \cdots \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})}, \cdots\right)\end{aligned} \quad (3.6)$$

$$e_t = (K\ U_{\mathrm{key}} + X_t\ U_{\mathrm{query}})u_{\mathrm{energy}}$$

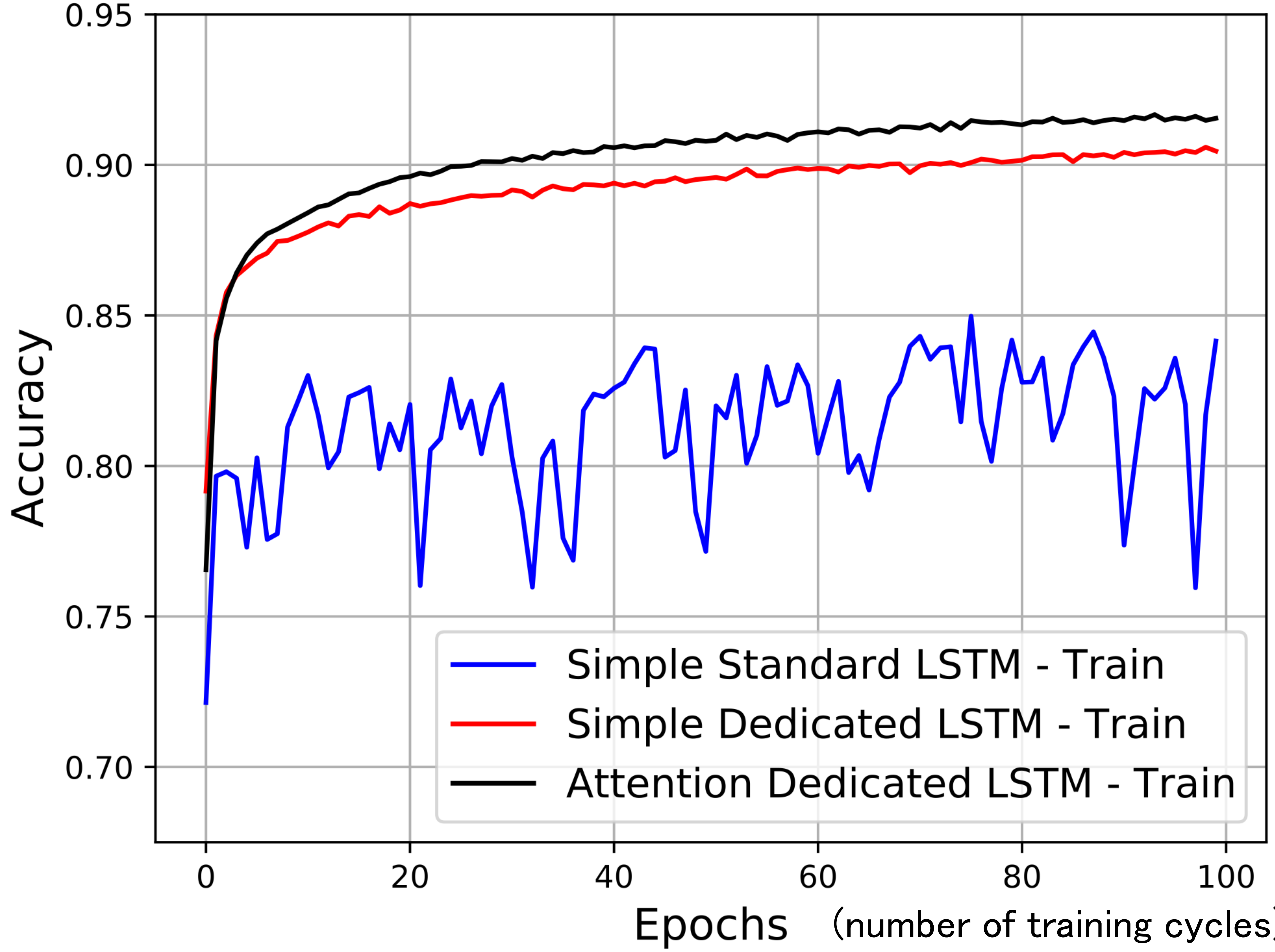






Performance of the custom network

Comparison to standard structure



(number of training cycles)

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

TP: true positive
TN: true negative
FP: false positive
FN: false negative

Improvement seen by using custom LSTM structure (red) and attention (black)

補足説明4

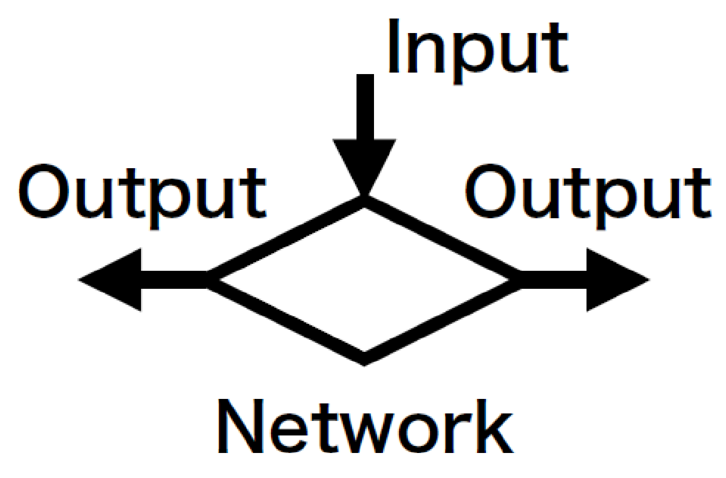
- Seedを正しく選ぶことが重要 (初段の改善)
  – Trainingの時はMC truthで正しいものだけを利用
- 順番は毎回シャッフルしている
  – 普通のLSTMではこれでダメになる
- 他のやり方
  – Vertexの情報をすべて含んだような「データベース」を作る
  → 各trackがどこに結びつけられるか分類する
  (transformer的な?まだ具体的に練られていない)
    - データベース自体がそのまま後段(flavor tagging)に使える?

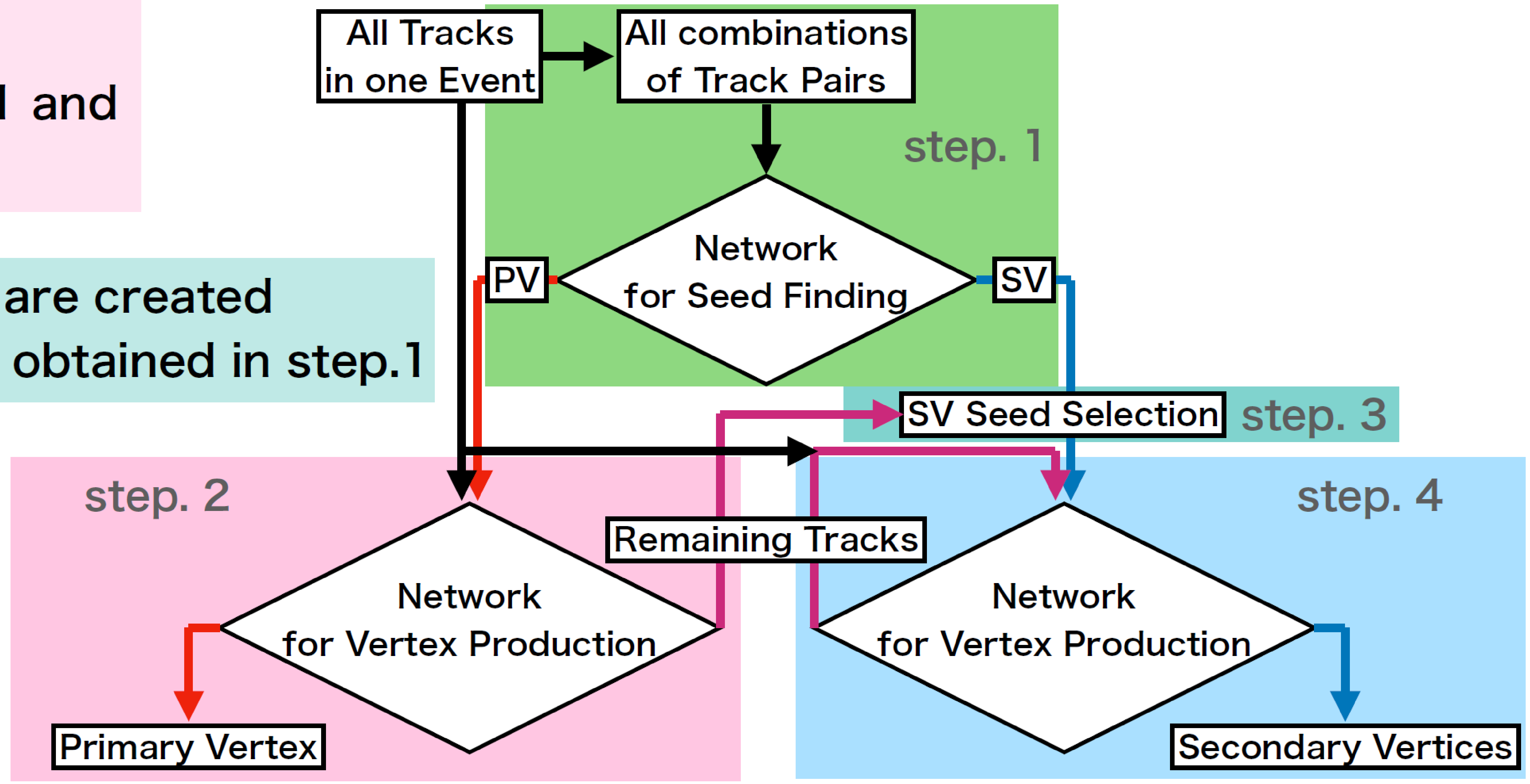
Vertex Finder

Algorithm for Vertex Finder

- Finding the vertices using following steps

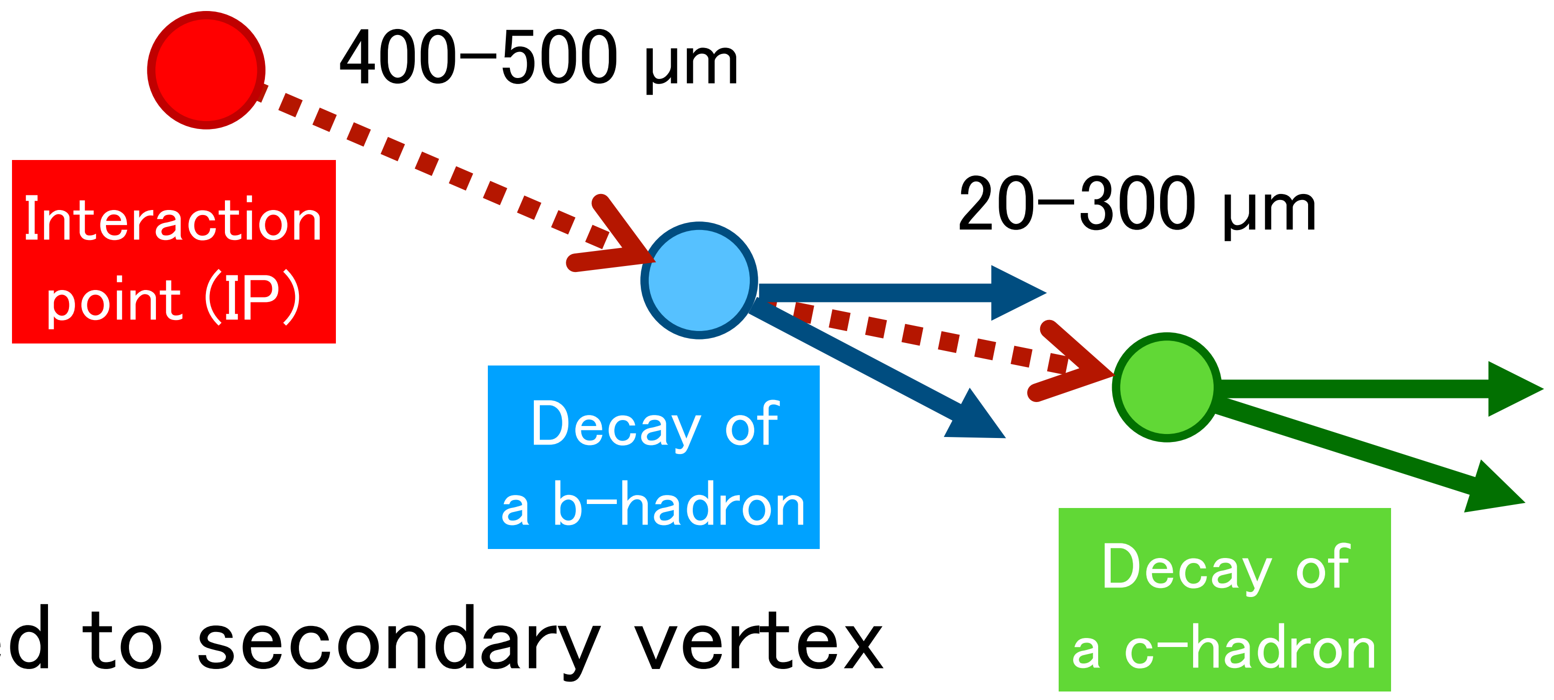
1. Considering all combinations of two tracks in a event, the vertex seeds are searched by “network for seed finding”
2. The primary vertex is created using the seeds of primary vertex obtained in step.1 and “network for vertex production”
3. The purer set of seeds of secondary vertices are created by screening the seeds of secondary vertices obtained in step.1
4. The secondary vertices are created using the seeds of secondary vertices selected in step.3 and “network for vertex production”





Performance of the DL-based vertex finder

Comparison with LCFIPlus (track-by-track criteria) with **bb samples**



- True label

1. Primary – tracks with no (semi)stable parents
2. Bottom – tracks originated from b-hadrons
3. Charm – tracks originated from c-hadrons
4. Others – other tracks (mainly V_0 tracks)

- Criteria

1. In secondary vertex – associated to secondary vertex
2. – of same decay chain – **all tracks in the vertex** from the same b parent
3. – of same parent – **all tracks in the vertex** from the same immediate parent (ie. success of b-c separation)

Performance of DL-based vertex finder

Criteria / True label	Primary	Bottom	Charm	Others
All tracks	307 657	187 283	180 143	42 888
In secondary vertex	2.2%	63.3%	68.4%	9.5%
– of same decay chain		62.3%	67.2%	
– of same parent		38.1%	36.2%	6.4%

Performance of LCFIPlus vertex finder

Criteria / True label	Primary	Bottom	Charm	Others
All tracks	307 657	187 283	180 143	42 888
In secondary vertex	0.2%	57.9%	60.3%	0.5%
– of same decay chain		57.5%	59.9%	
– of same parent		34.0%	37.2%	0.3%

- 5-10% higher efficiency on the reconstruction of secondary vertices
- More contamination of primary and other tracks
  (need additional selection on track quality etc.)

Adaptation to C++ / LCFIPlus / ILCSoft (Marlin)

Method for inference (evaluation) in C++

Training in python
Inference in C++

- Tensorflow/Keras is used for building/training the network
  - Fully python (version 3)
  - We obtain input with LCIO → ROOT tree → NumPy conversion
- LCFIPlus is running as a Marlin processor, fully C++
  - For comparison of the flavor tagging, the output vertices should be in LCIO or LCFIPlus format.

- Keras is provided only in python, but Tensorflow has official C++ implementation
  - This is one reason we chose Tensorflow as the framework (while PyTorch only has beta implementation on C++ port).
  - Inference (evaluation of the network) can be done without Keras, thus possible to run in C++.

- We introduce VertexFinderwithDL algorithm inside LCFIPlus (thus possible to be called from Marlin)
  - Tensorflow and bazel (as dependency) are needed to be installed
  - Can run both with GPU and without GPU (cuda / cuDNN necessary for GPU run)
  - Results have compared with python version; identical result obtained
  - Output vertices are compatible with LCFIPlus output
    - Vertex fitting (to obtain position, χ^2 etc.) is done using LCFIPlus functions after selecting tracks with DL networks.

CMake results

```
-- Found LCFIVertex: /gluster/data/ilc/ilcsoft/v02-02/LCFIVertex/v00-08
-- Found Tensorflow: /home/goto/local/include/tf ◀
-- Found Protobuf: ◀
-- Found Eigen3: /home/goto/local/include/eigen3 (Required is at least version "2.91.0") ◀
-- Check for ROOT_CINT_EXECUTABLE: /gluster/data/ilc/ilcsoft/v02-02/root/6.18.04/bin/rootcint
```

Adaptation to C++ / LCFIPlus / ILCSoft

```
2020-11-14 21:33:38.248302: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.1
2020-11-14 21:33:38.248381: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10
2020-11-14 21:33:38.248490: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcufft.so.10
2020-11-14 21:33:38.248562: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcurand.so.10
2020-11-14 21:33:38.248660: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusolver.so.10
2020-11-14 21:33:38.248731: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusparse.so.10
2020-11-14 21:33:38.248835: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
2020-11-14 21:33:38.251605: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1697] Adding visible gpu devices: 0, 1
2020-11-14 21:33:38.251752: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1096] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-11-14 21:33:38.251874: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102]      0 1
2020-11-14 21:33:38.251999: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] 0:   N Y
2020-11-14 21:33:38.252111: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] 1:   Y N
2020-11-14 21:33:38.254091: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1241] Created TensorFlow device (/job:localhost/replica:0/task:0/devi
0, name: TITAN RTX, pci bus id: 0000:81:00.0, compute capability: 7.5)
2020-11-14 21:33:38.255119: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1241] Created TensorFlow device (/job:localhost/replica:0/task:0/devi
1, name: TITAN RTX, pci bus id: 0000:c1:00.0, compute capability: 7.5)
2020-11-14 21:33:38.334339: I tensorflow/cc/saved_model/loader.cc:203] Restoring SavedModel bundle.
2020-11-14 21:33:38.446862: I tensorflow/cc/saved_model/loader.cc:152] Running initialization op on SavedModel bundle at path: /home/goto/ILC/Deep_L
6_50000samples_100epochs_ps_100epochs_s
2020-11-14 21:33:38.509773: I tensorflow/cc/saved_model/loader.cc:333] SavedModel load for tags { serve }; Status: success: OK. Took 292443 microsec
2020-11-14 21:33:44.005873: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10
[ MESSAGE "Marlin"]  ---- no GEAR XML file given  ---------
[ MESSAGE "VertexFindingwithDL"]
[ MESSAGE "VertexFindingwithDL"] ---- VertexFindingwithDL -  parameters:
[ MESSAGE "VertexFindingwithDL"]         Algorithms:  VertexFindingwithDL
[ MESSAGE "VertexFindingwithDL"]         IgnoreLackOfVertexRP:  0
[ MESSAGE "VertexFindingwithDL"]         MCPCollection:  MCParticlesSkimmed
[ MESSAGE "VertexFindingwithDL"]         MCPFORelation:  RecoMCTruthLink
[ MESSAGE "VertexFindingwithDL"]         MagneticField:  3.5
[ MESSAGE "VertexFindingwithDL"]         PFOCollection:  PandoraPFOs
[ MESSAGE "VertexFindingwithDL"]         PIDAlgorithmName:  LikelihoodPID
[ MESSAGE "VertexFindingwithDL"]         PrintEventNumber:  0
[ MESSAGE "VertexFindingwithDL"]         ReadSubdetectorEnergies:  1
[ MESSAGE "VertexFindingwithDL"]         TrackHitOrdering:  1
[ MESSAGE "VertexFindingwithDL"]         UpdateVertexRPDaughters:  0
[ MESSAGE "VertexFindingwithDL"]         UseMCP:  1
[ MESSAGE "VertexFindingwithDL"] -------------------------------------------------
```

Libraries for GPU

Setup GPUs

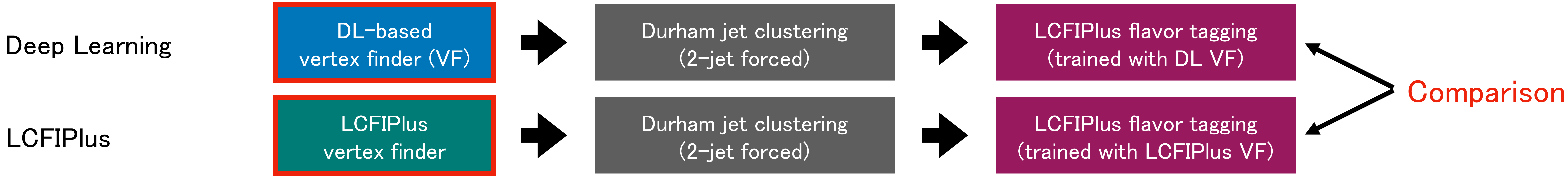
Restoring model

My Processor "Vertex Finder with DL" is running

Performance of the flavor tagging (FT)

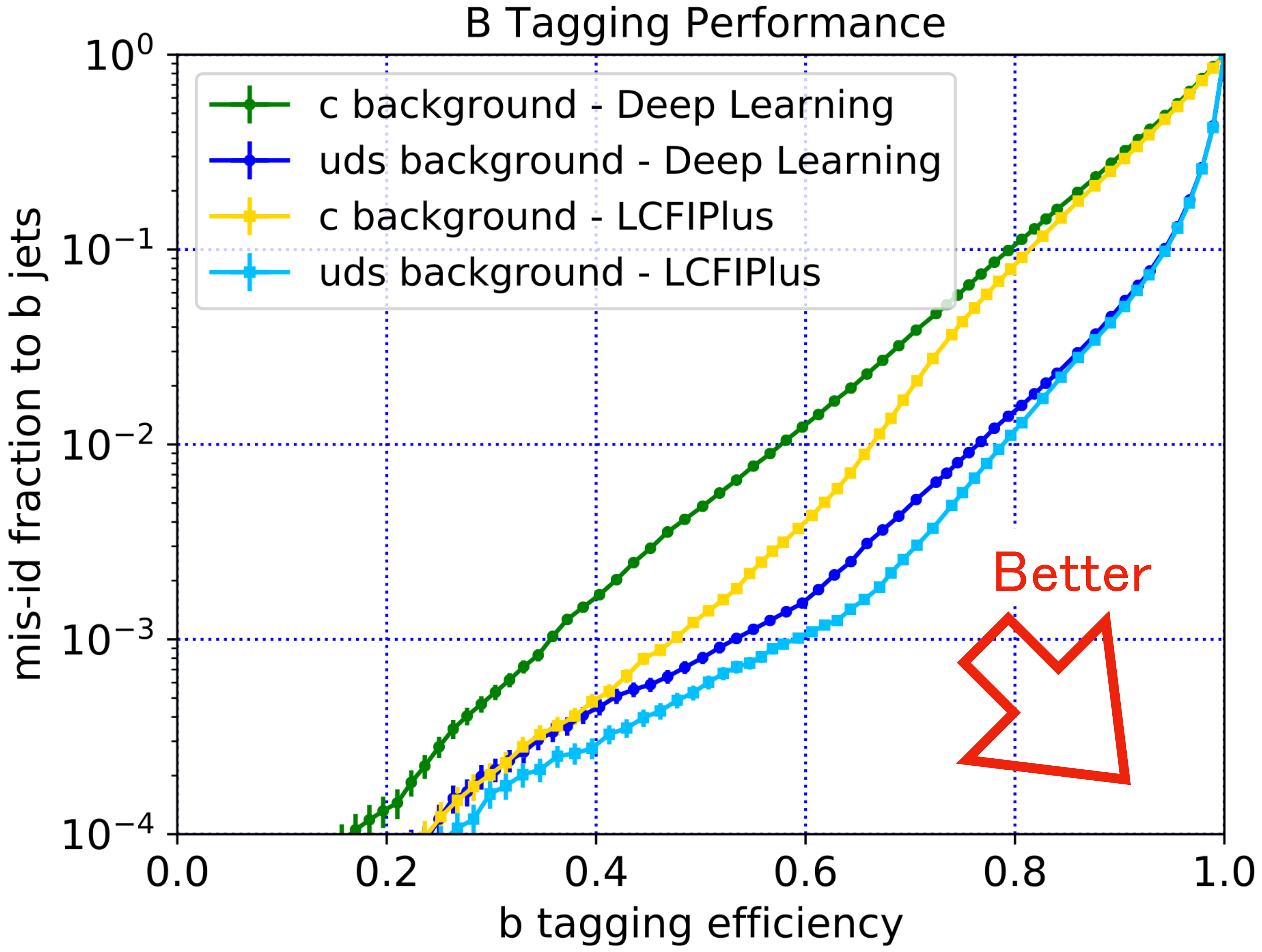
Procedure for flavor tagging

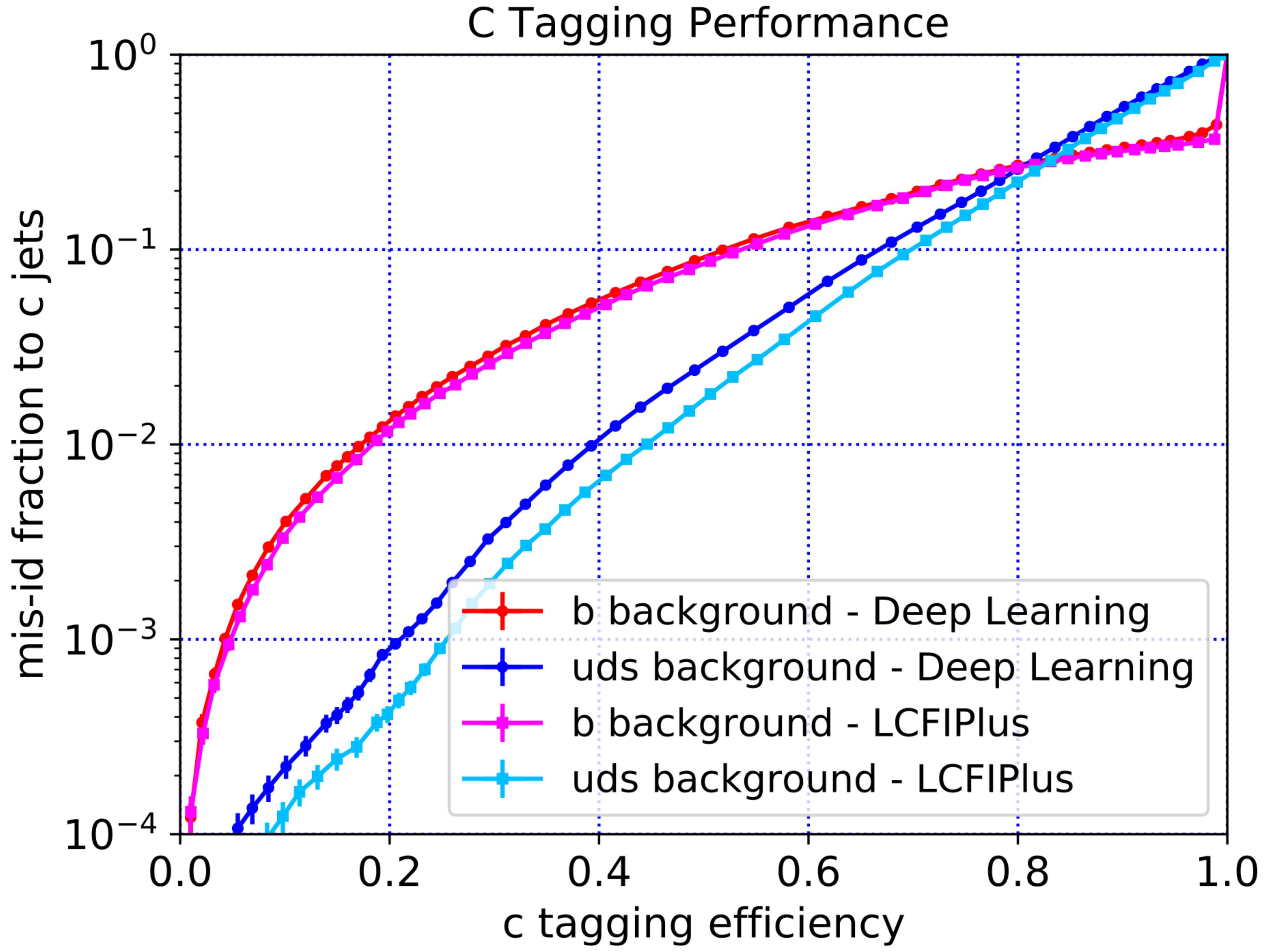
- Replace vertex finder and use other algorithm as same as LCFIPlus' case (with same parameter)



- The performance of b-tagging of LCFIPlus cannot be reproduced with DL vertex finder
  - Similar performance in c-tagging
  - Probably due to contamination of primary tracks
  - Tuning of parameters / input variables are highly optimized with LCFIPlus → some bias on LCFIPlus
- DL-based vertex finder has an advantage of possibility of closer connection to flavor tagging algorithm
  - "organic" connection of networks possible if FT fully written in DNN

→ FT algorithm to be rewritten with DNN





まとめ

- 深層学習を用いた二次崩壊点検出に挑戦した
  - 基本的には、既存のalgorithm (LCFIPlus)の思想を踏襲
    - 比較がしやすい
  - 崩壊点検出はfitting結果を用いたパターン分類
    - 機械学習の限界 (or可能性)を探るため、
できるだけ既存の解析手法(fitter)に頼らないようにしている
      - 交差点の探索をnetwork化できれば汎用的に使えるかもしれない
      - もちろんfitterを組み合わせていく方法も考えられる
(軽く試したところ、若干良くなる程度だった)
  - まだ改善が必要 (contaminationを減らさないといけない)
- 二次崩壊点検出とフレーバー識別を組み合わせたネットワークをどう構築するか → LCFIPlusの置き換えへ

Backup

Summary and Prospects

Source codes:
https://github.com/Goto-K/VertexFinderwithDL (python part)
https://github.com/Goto-K/LCFIPlus (adaptation to LCFIPlus)

Summary

- We developed a vertex finding algorithm based on modern deep neural networks.
- Track association done with customized RNN-type network with attention mechanism.
- Efficiency of the reconstruction of secondary vertices is improved,
  while mis-reconstruction of primary / other tracks to secondary vertices is increased.
- Inference process in C++ incorporated to iLCSoft/LCFIPlus (with C++ version of Tensorflow) has been developed.
- Cannot reproduce performance of LCFIPlus in b-tagging (with similar performance in c-tagging).
  Replacing flavor tagging with full DNN and closer connection of VF and FT networks may improve the situation.
- Paper under ILD review (excl. flavor tagging performance).

DL-based vertex finder

Criteria / True label	Primary	Bottom	Charm	Others
All tracks	307 657	187 283	180 143	42 888
In secondary vertex	2.2%	63.3%	68.4%	9.5%
– of same decay chain		62.3%	67.2%	
– of same parent		38.1%	36.2%	6.4%

LCFIPlus vertex finder

Criteria / True label	Primary	Bottom	Charm	Others
All tracks	307 657	187 283	180 143	42 888
In secondary vertex	0.2%	57.9%	60.3%	0.5%
– of same decay chain		57.5%	59.9%	
– of same parent		34.0%	37.2%	0.3%

Prospects

- Improvement of the vertex finder
  - More tuning of the “seed finding” network, using more appropriate network to use “crossing point” of two tracks
  - Including more physical properties to RNN network as well as improving structure
- Development of DNN-based flavor tagging
  - More input variables, more layers
  - Non-simple connection of VF and FT network, eg. connecting hidden layer of two networks
    (discussion with AI experts ongoing in regular communication)

This work is supported by RCNP (Osaka U.) Project
“Application of deep learning to accelerator experiments”.

3. Inference with C++

For Evaluation in LCFIPlus

- I want to show the performance of Flavor Tagging with my Vertex Finder
  ➡ I need to run these networks in LCFIPlus

- I completed the implementation the Vertex Finder with DL (Tensorflow 2.1.0) to the LCFIPlus in iLCSoft (v02-02)
  - Some cmake files are required and some find packages are added to the CMakeLists
  - Also I have to use the shared libraries of tensorflow C++ API built by "bazel"

CMake results

```
-- Found LCFIVertex: /gluster/data/ilc/ilcsoft/v02-02/LCFIVertex/v00-08
-- Found Tensorflow: /home/goto/local/include/tf
-- Found Protobuf:
-- Found Eigen3: /home/goto/local/include/eigen3 (Required is at least version "2.91.0")
-- Check for ROOT_CINT_EXECUTABLE: /gluster/data/ilc/ilcsoft/v02-02/root/6.18.04/bin/rootcint
-- Check for ROOT_DICT_OUTPUT_DIR: /home/goto/ILC/LCFIPlus/build/rootdict
-- Check for ROOT_DICT_CINT_DEFINITIONS:
-- Found Doxygen: /usr/bin/doxygen (found version "1.8.14") found components:  doxygen dot
--
-- ------------------------------------------------------------------------------
```

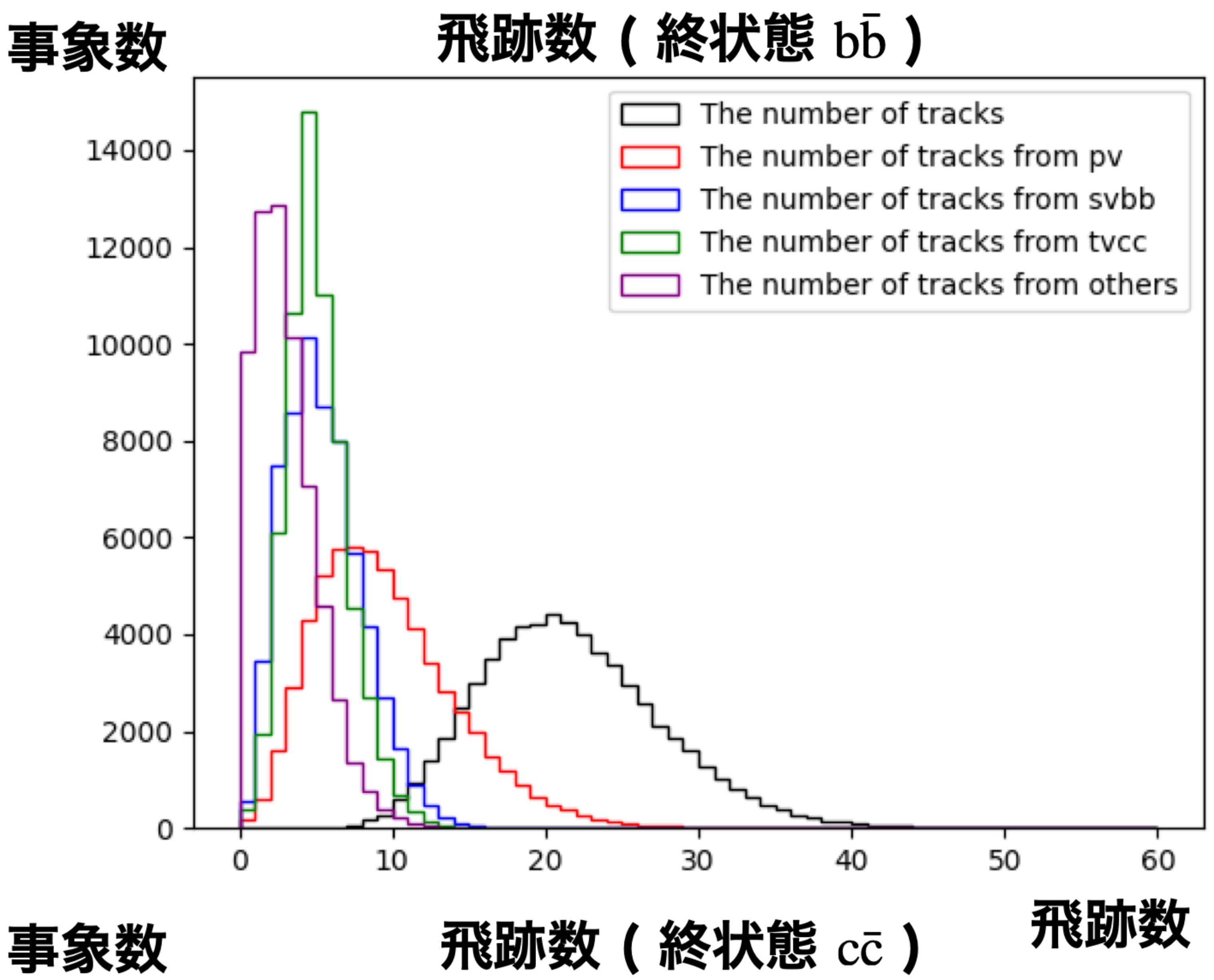
3. Inference with C++

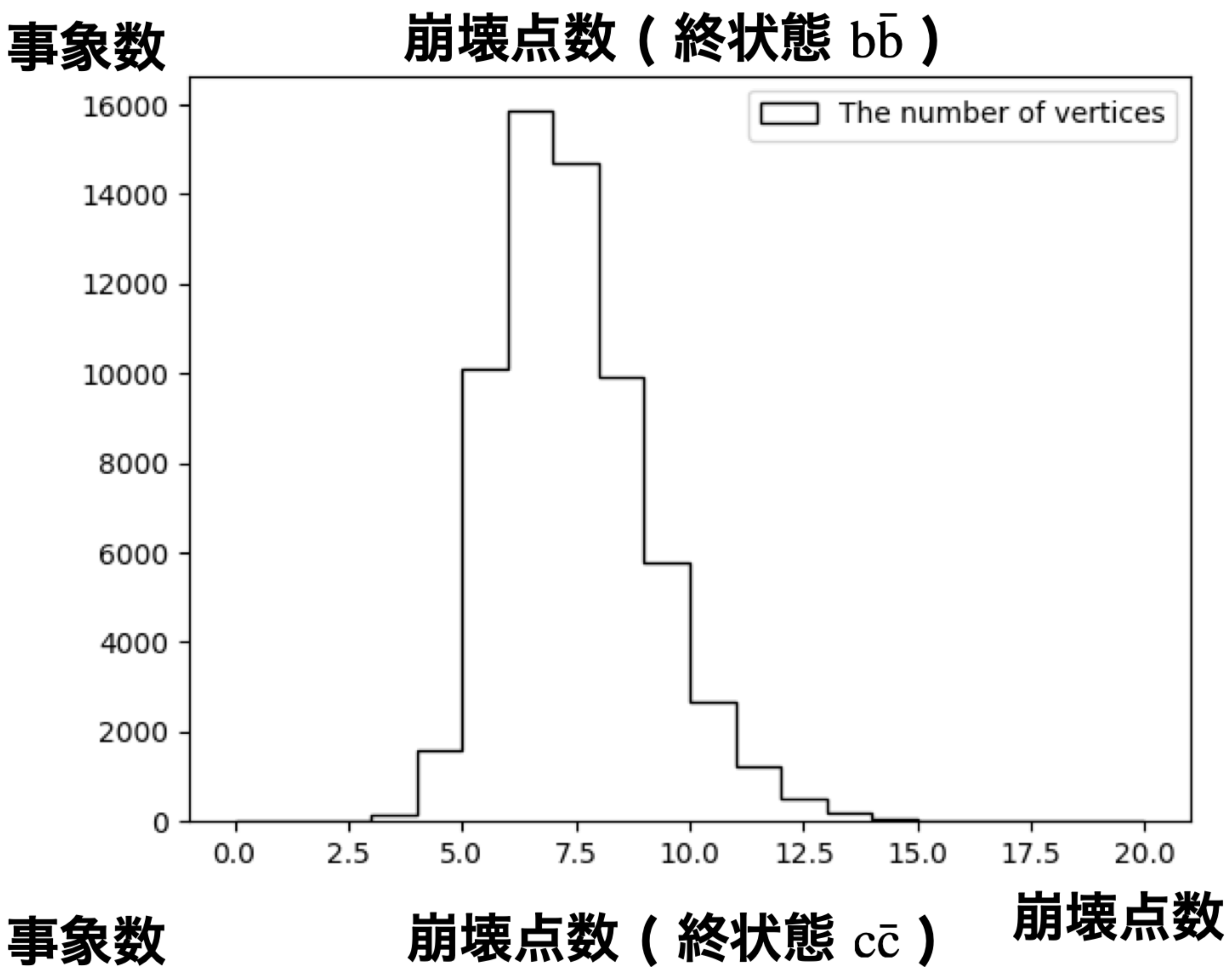
Software setups @ “beep-gpu” server in Kyushu Univ

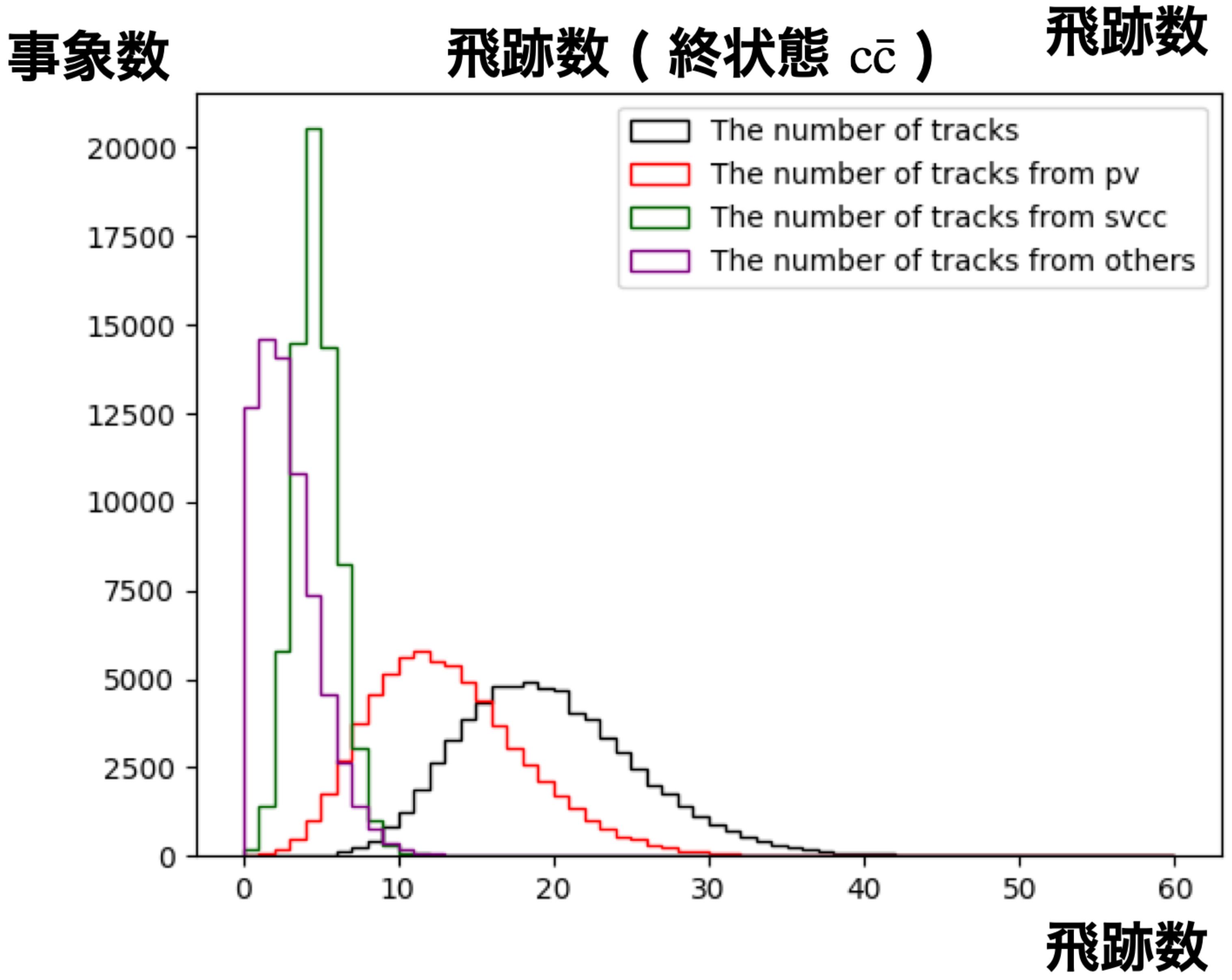
- For use Tensorflow in LCFIPlus (iLCSoft)
  - Download Tensorflow from GitHub
  - Install Bazel v0.29.1
  - Build Tensorflow C++ API & make shared library (libtensorflow_cc.so, libtensorflow_framework.so)
    - Tensorflow v2.1.0 / CUDA v10.1 / cuDNN v7 / Eigen v3.3.90 / Protobuf v3.8 /
      (g++ v8.4.0 / C++11, 14)
  - Move header files and libraries to the /usr/local/include/tf and …/lib/ or your own local
    - Also need to put the eigen3/unsupported, google/protobuf, tf/absl in the /usr/local/include/
  - Make cmake file (FindTensorflow, Eigen3, Protobuf) and write find_package in CMakeLists.txt
    - Include/eigen3/unsupported and libtensorflow_framework.so are not available in this way
      We have to use the absolute path to these files
- Install iLCSoft v02-02 (please give attention to cmake version)

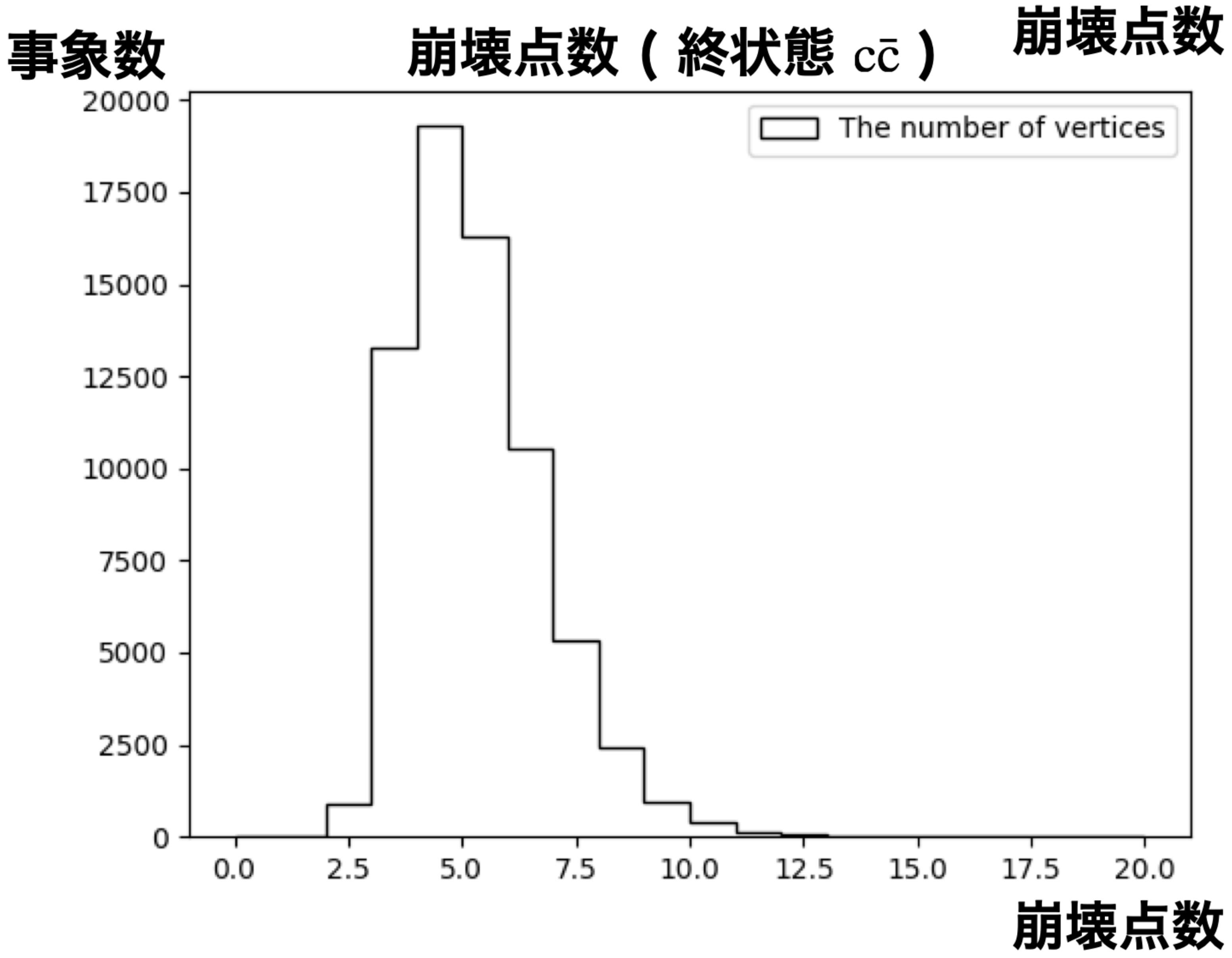
1. イントロダクション

データの性質





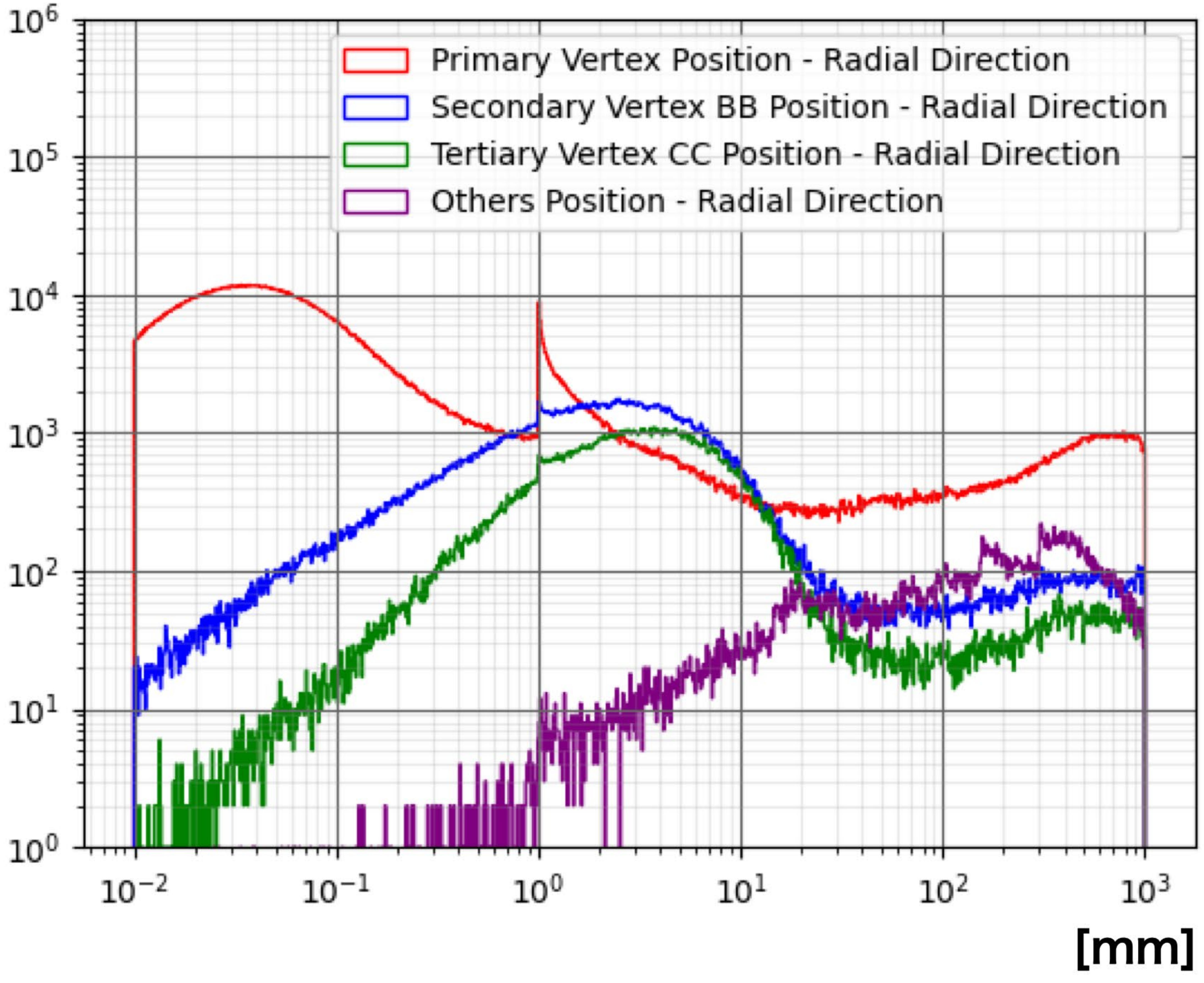




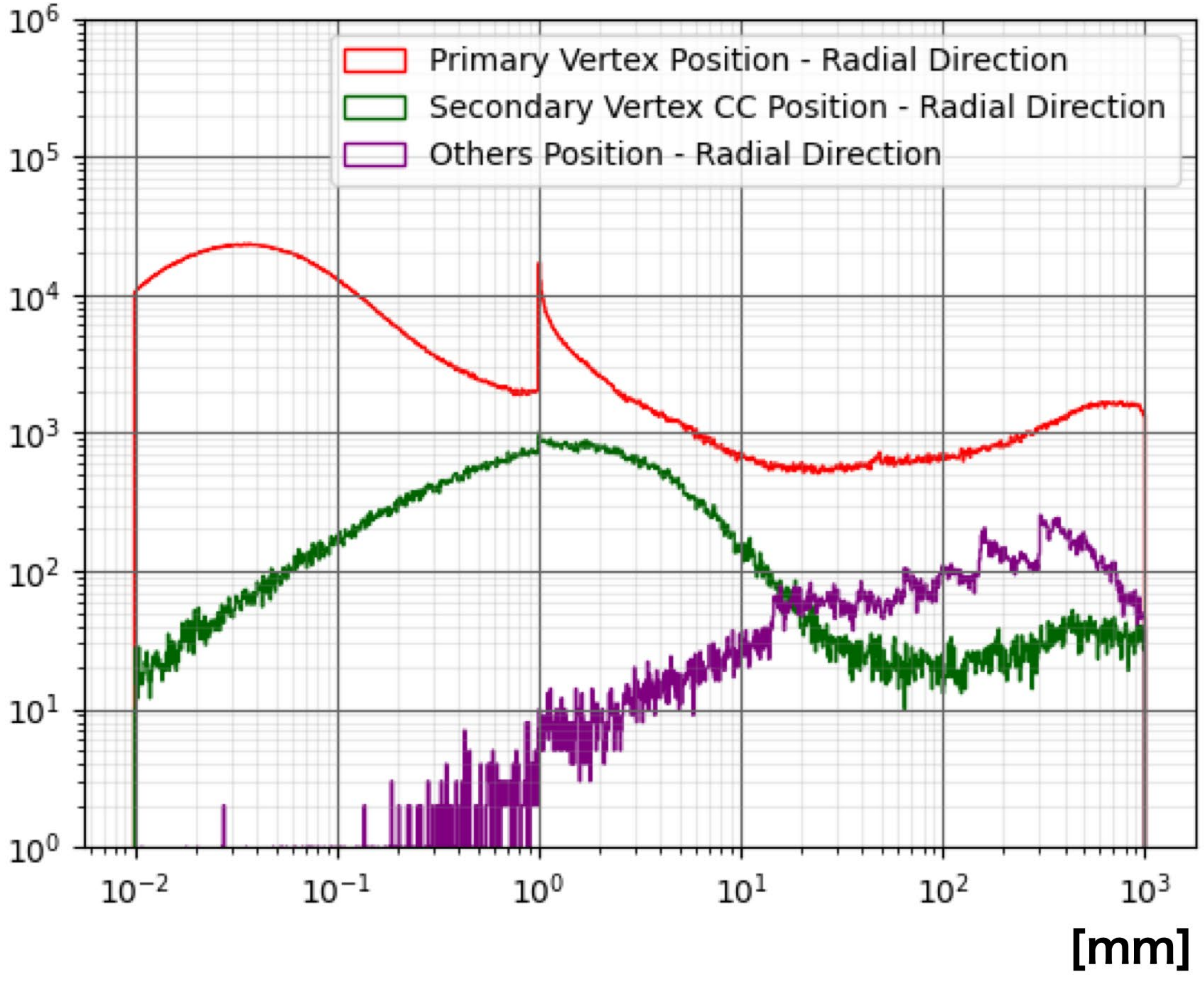
1. イントロダクション

データの性質

終状態 $b\bar{b}$



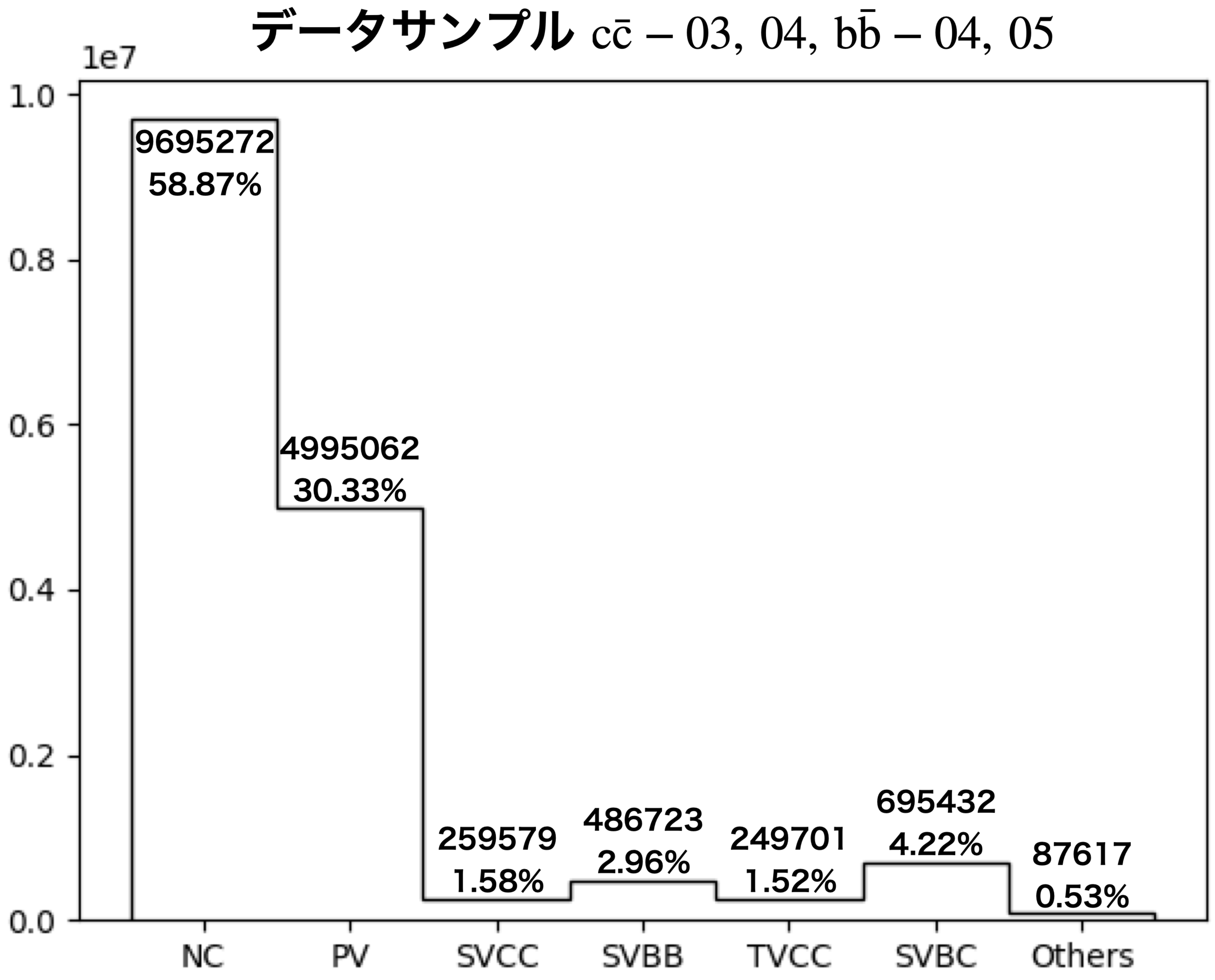
終状態 $c\bar{c}$



1. イントロダクション

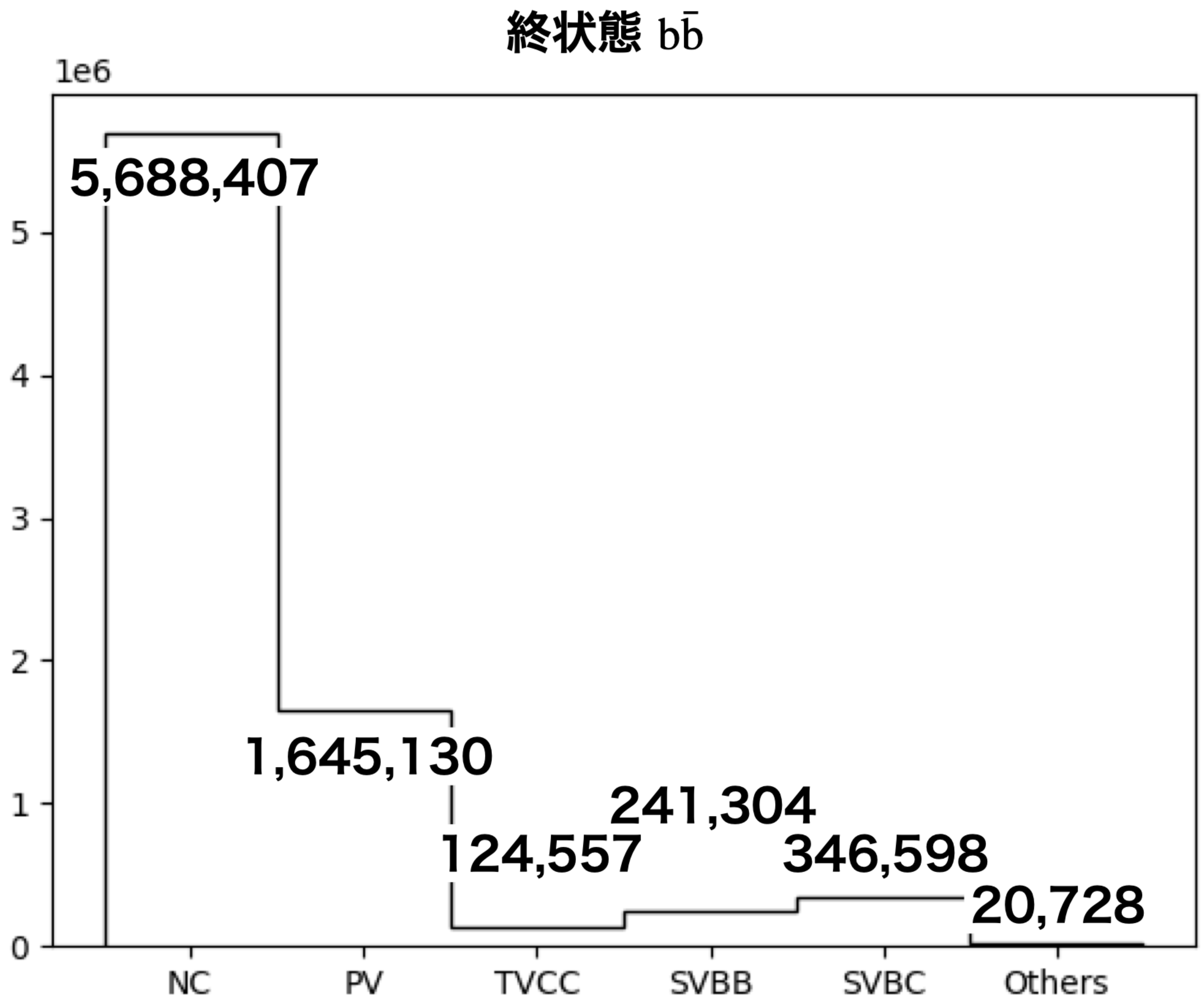
データの性質

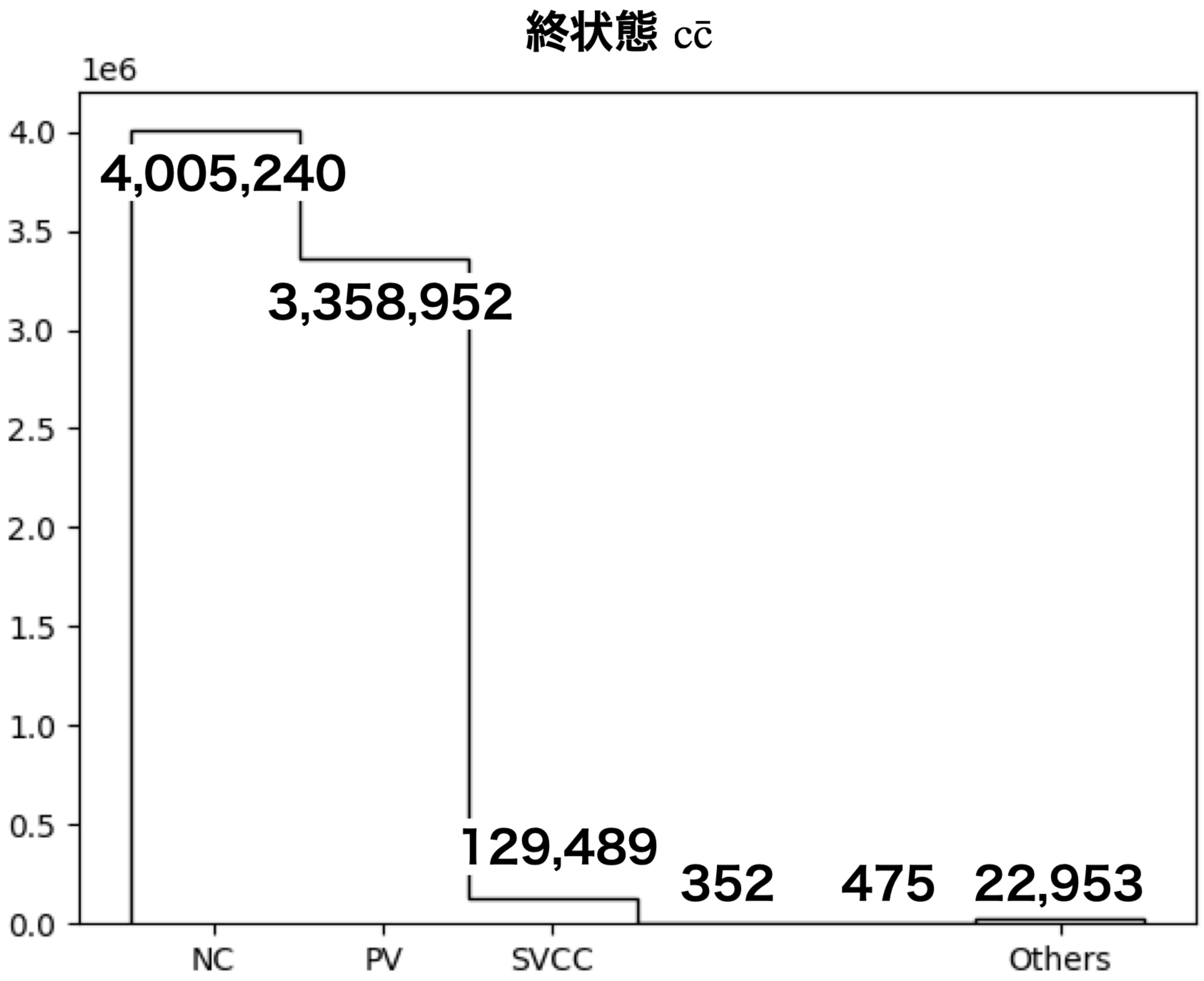
- 飛跡対 (崩壊点の種) の種類
  - NC : 結合していない飛跡対
  - PV : primary vertex 由来
  - SVCC : 終状態 $c\bar{c}$ での secondary vertex 由来
  - SVBB : 終状態 $b\bar{b}$ での secondary vertex 由来
  - TVCC : 終状態 $b\bar{b}$ での tertiary vertex 由来
  - SVBC : 終状態 $b\bar{b}$ での secondary vertex から1本
    tertiary vertex から1本で構成された飛跡対
  - Others : その他の崩壊点由来の飛跡対
    - V^0の崩壊, 光子変換など



1. イントロダクション

データの性質

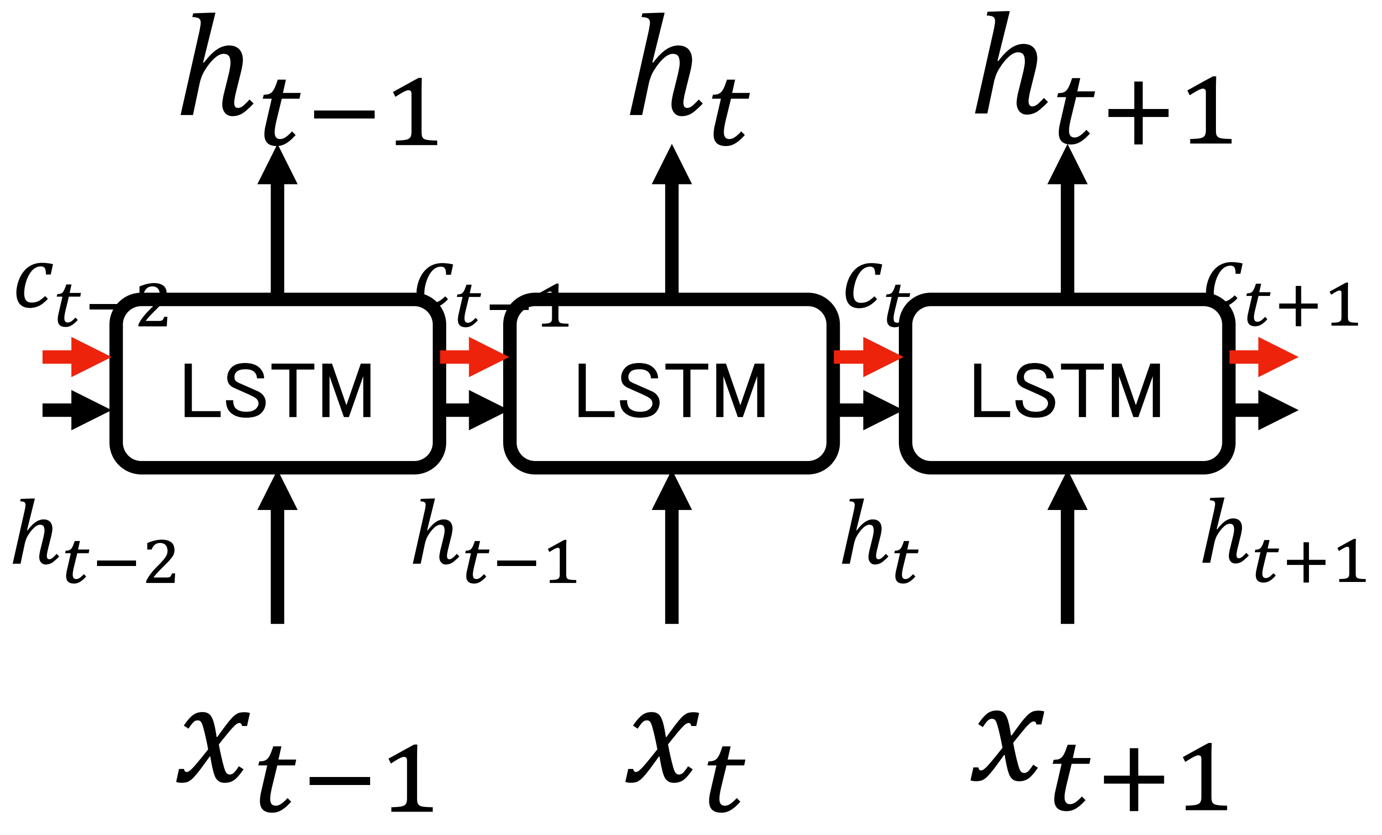




1. イントロダクション

LSTM (Long short−term memory)

- 日本語では「長短期記憶」
- リカレントニューラルネットワークの問題を解決する為に開発されたネットワーク
  - リカレントニューラルネットワークは長期的な情報を保持出来ない
  - 4つのゲートと記憶セルを持っている
    - ゲートは重み更新についての問題を解決するためのテクニック
    - 記憶セル c は長期的な情報保持のためのテクニック

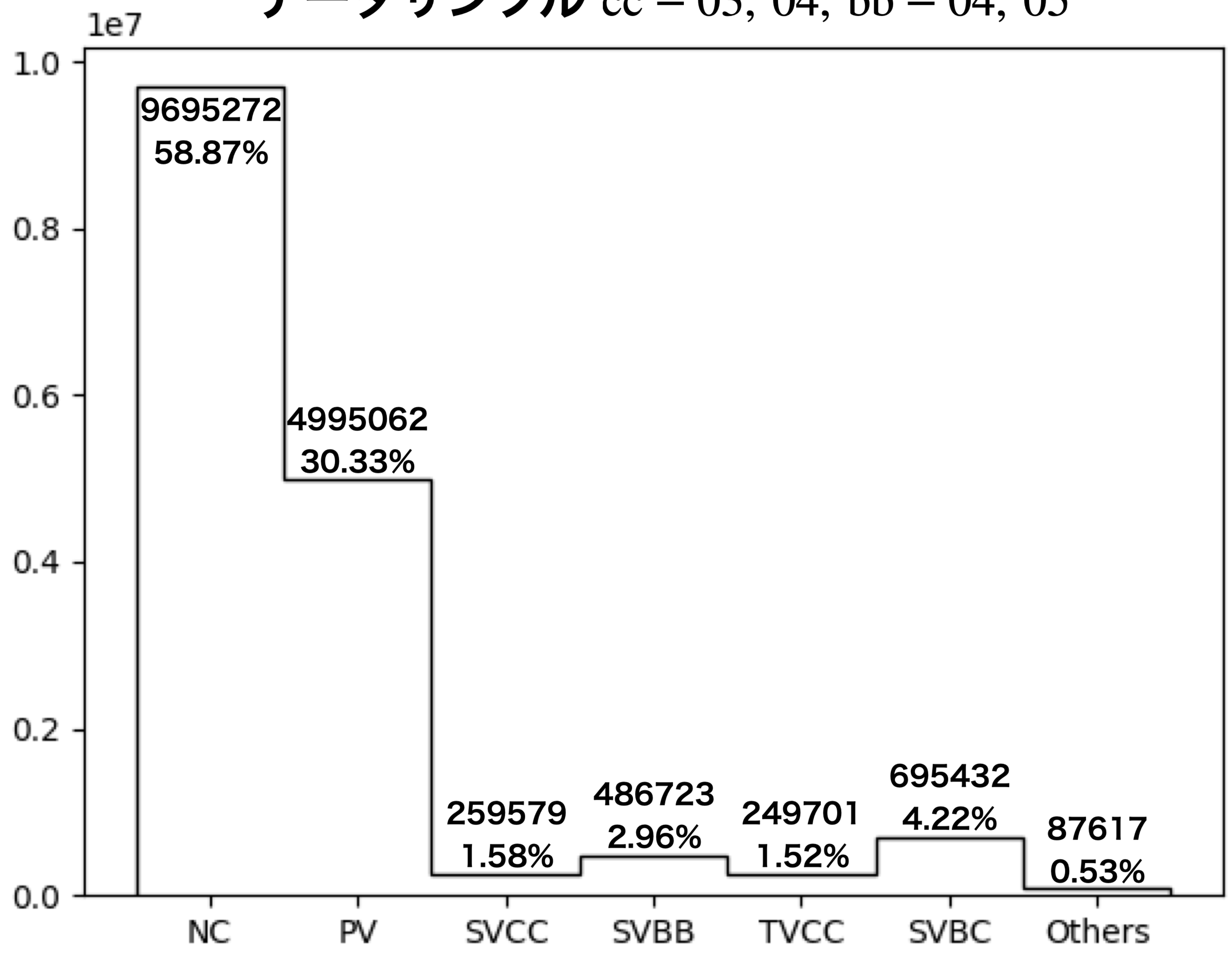


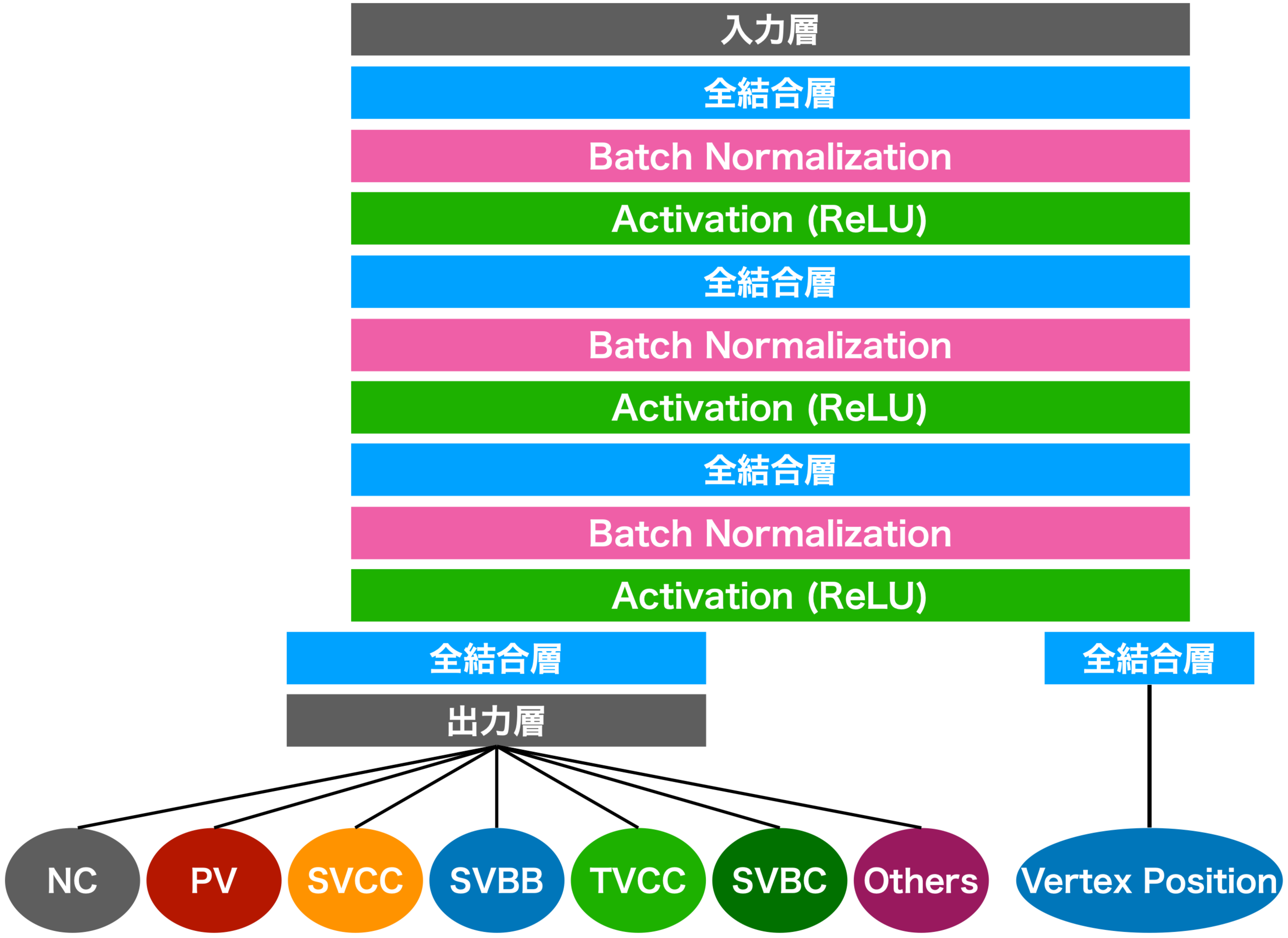
2. 崩壊点検出の為のニューラルネットワーク

飛跡対についてのネットワーク –構造と性能–

- シンプルなネットワークを使用
- 不均衡データである為、損失関数に重みを付ける
  - 損失関数 : 学習に使用する評価関数
    (最小化するように学習が進む)

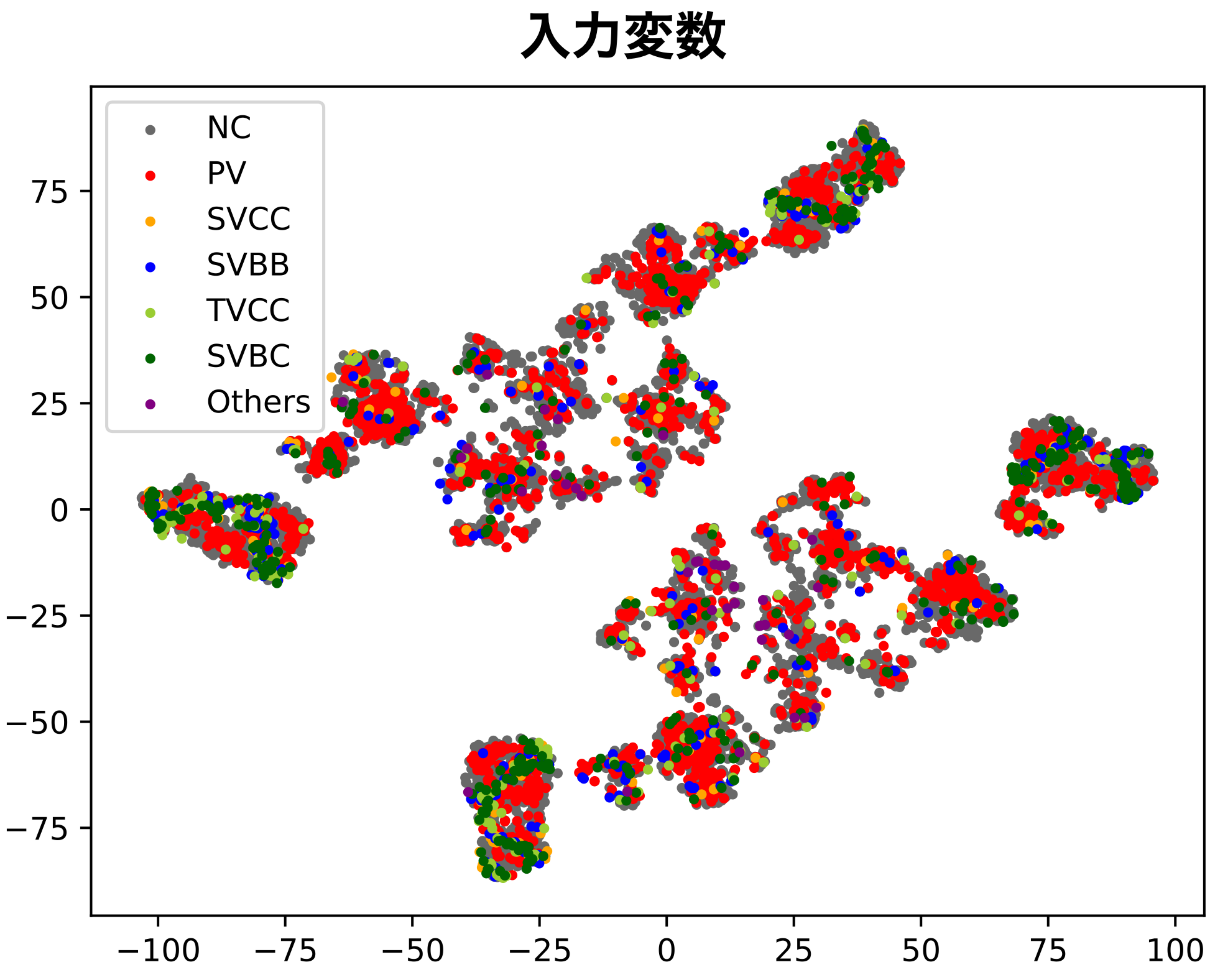
データサンプル $c\bar{c}$ – 03, 04, $b\bar{b}$ – 04, 05

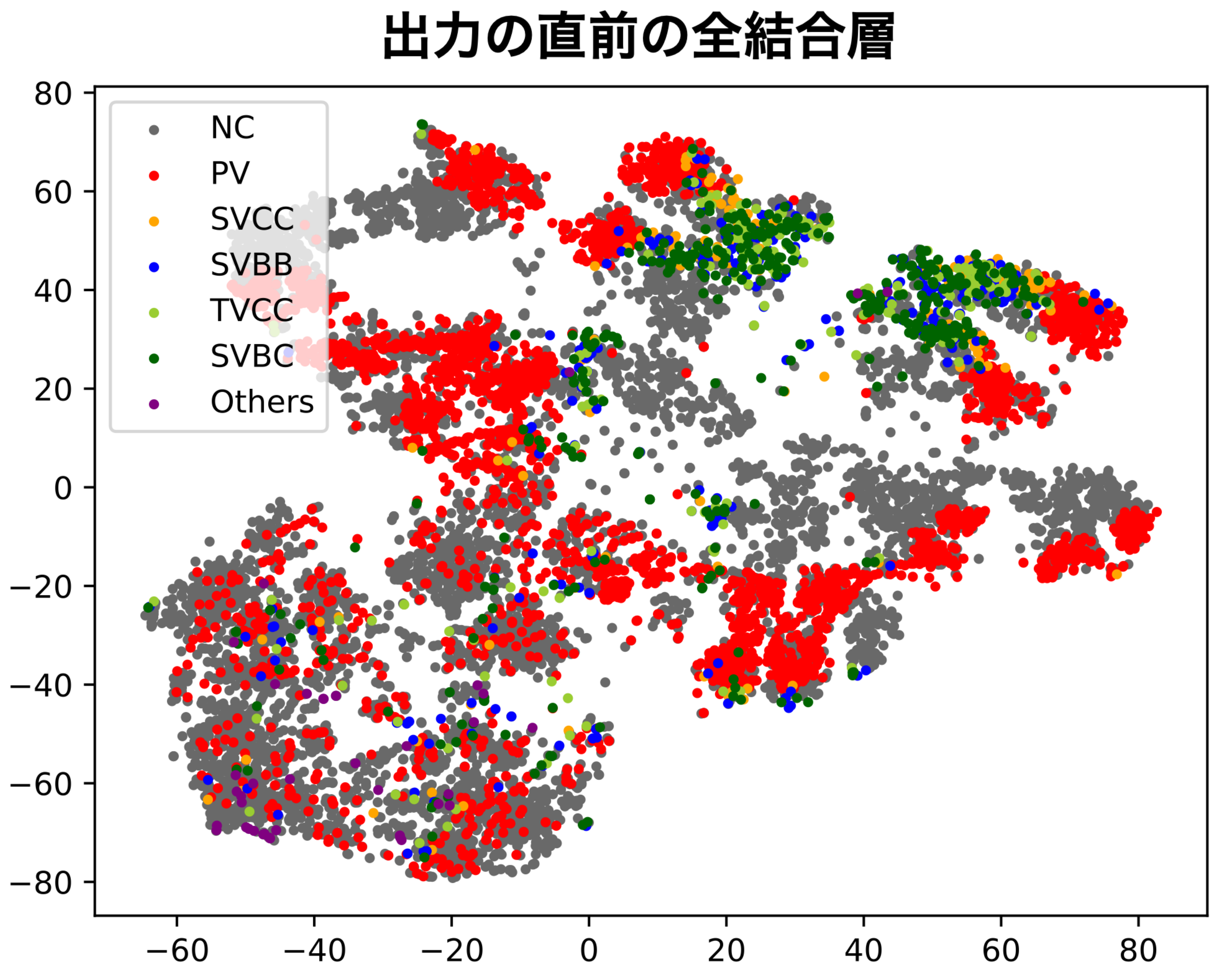




2. 崩壊点検出の為のニューラルネットワーク

飛跡対についてのネットワーク −構造と性能−





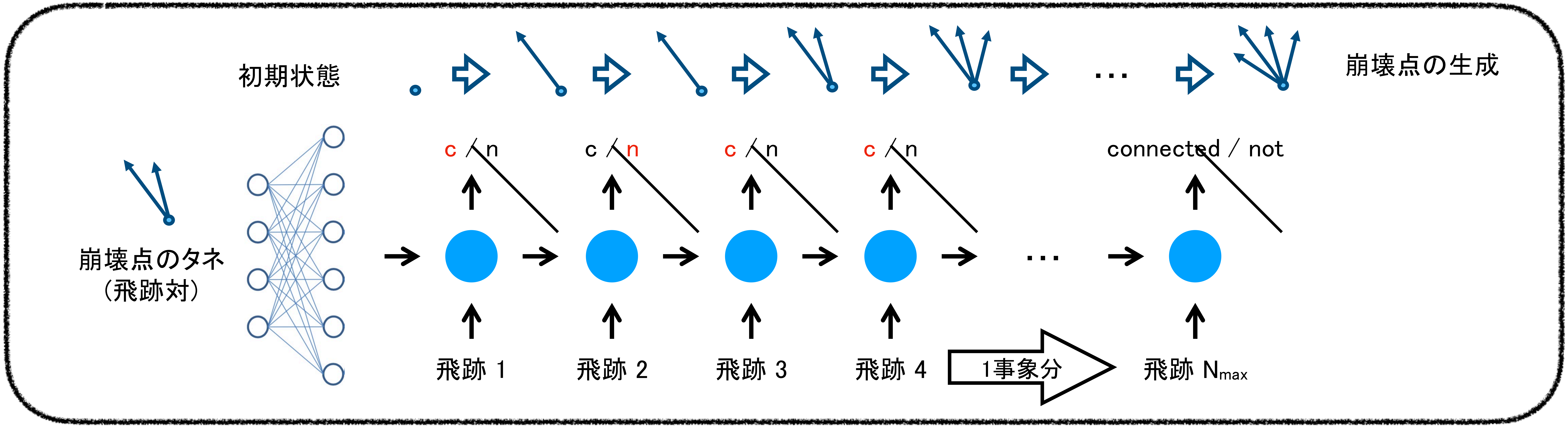
2. 崩壊点検出の為のニューラルネットワーク

LSTMを用いたアプローチ

◎ 二本以上の飛跡を処理できるネットワークを構築したい

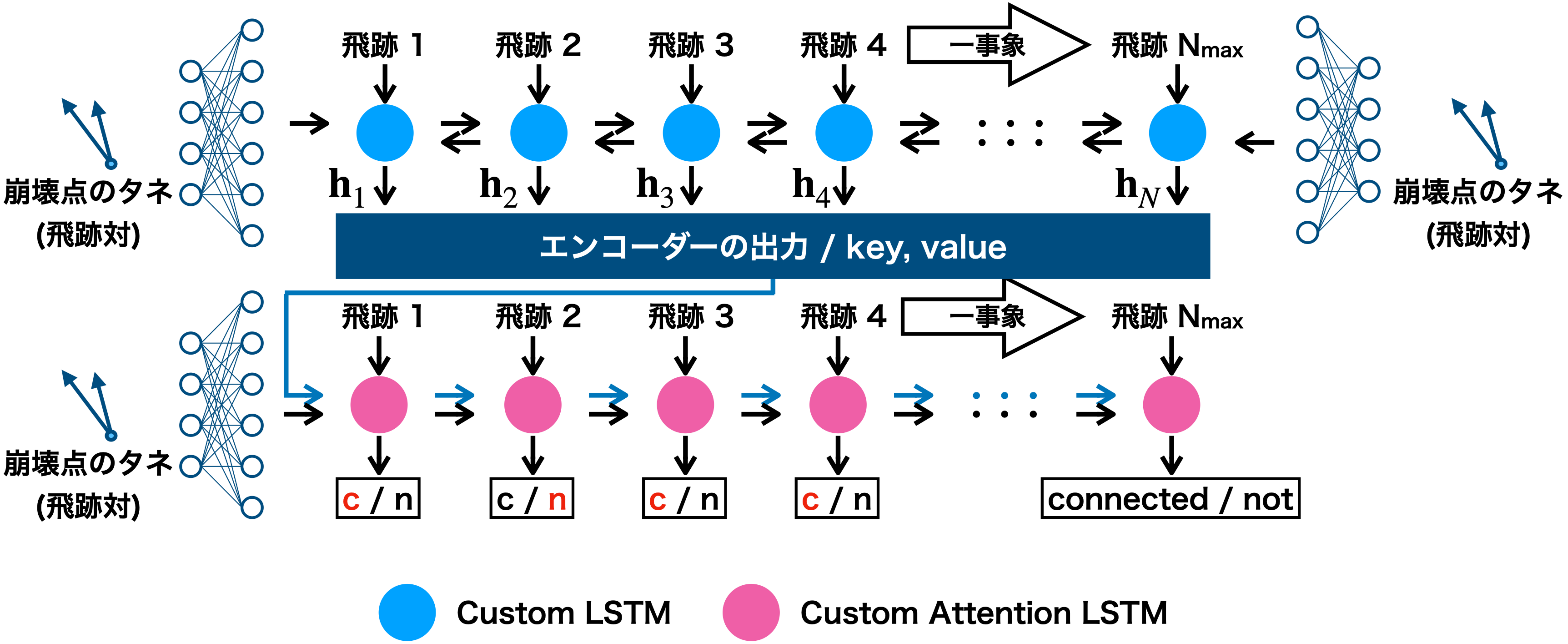
- 問題点
  - 含まれる飛跡の数が事象毎に異なる
  - 含まれる崩壊点の数が事象毎に異なる

  ➡ 可変長なネットワーク(リカレントニューラルネットワーク)
- 初期状態(崩壊点のタネ)に対して、飛跡が繋がっているかどうか (初期状態を学習する)



2. 崩壊点検出の為のニューラルネットワーク

任意の数の飛跡についてのネットワーク -構造-



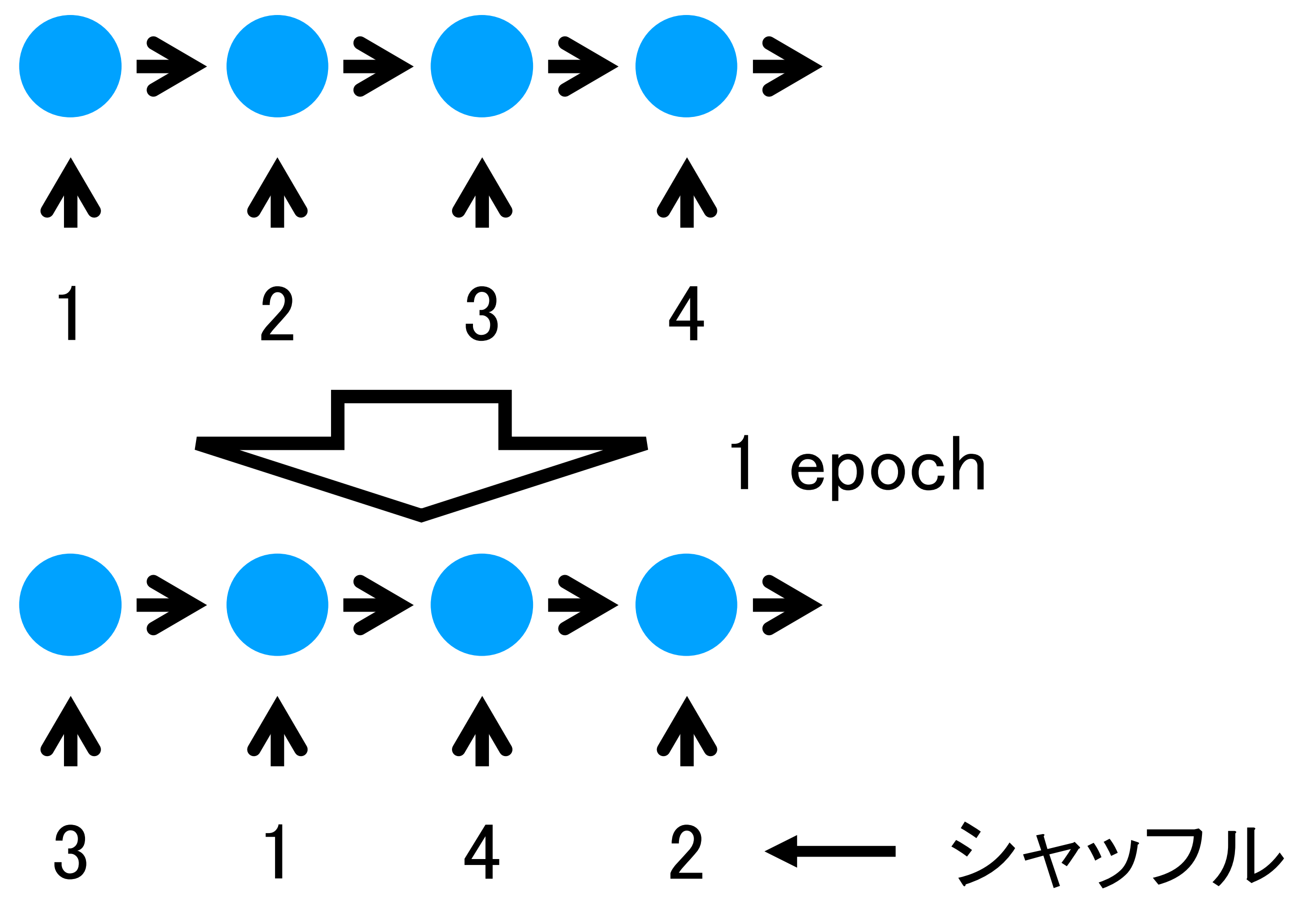
2. 崩壊点検出の為のニューラルネットワーク

任意の数の飛跡についてのネットワーク -学習と性能-

- 損失関数 : binary cross entropy
- 最適化/学習率 : Adam/0.001
  - 重み更新の手法とステップ幅
- 学習回数(Epoch) : 100 epochs
- バッチサイズ : 32
  - 重み更新毎のサンプル数
- Framework / Hardware : Tensorflow, Keras / TITAN RTX

- 20000 事象 (1159547 samples) ➡ 1 epoch毎にランダムに50000 sampleを選び学習
  - 崩壊点毎に教師データが1 sample生成される

- ゼロパディングとマスク
  - 学習時は全事象の飛跡の最大数で「ゼロ埋め (パディング)」し、学習に影響が出ないよう「マスク」している
- 飛跡順のシャッフル
  - 本来、飛跡に順序はない為、学習においても出来る限り系列に依存しないよう 1 epoch毎に飛跡の順序をシャッフルしている

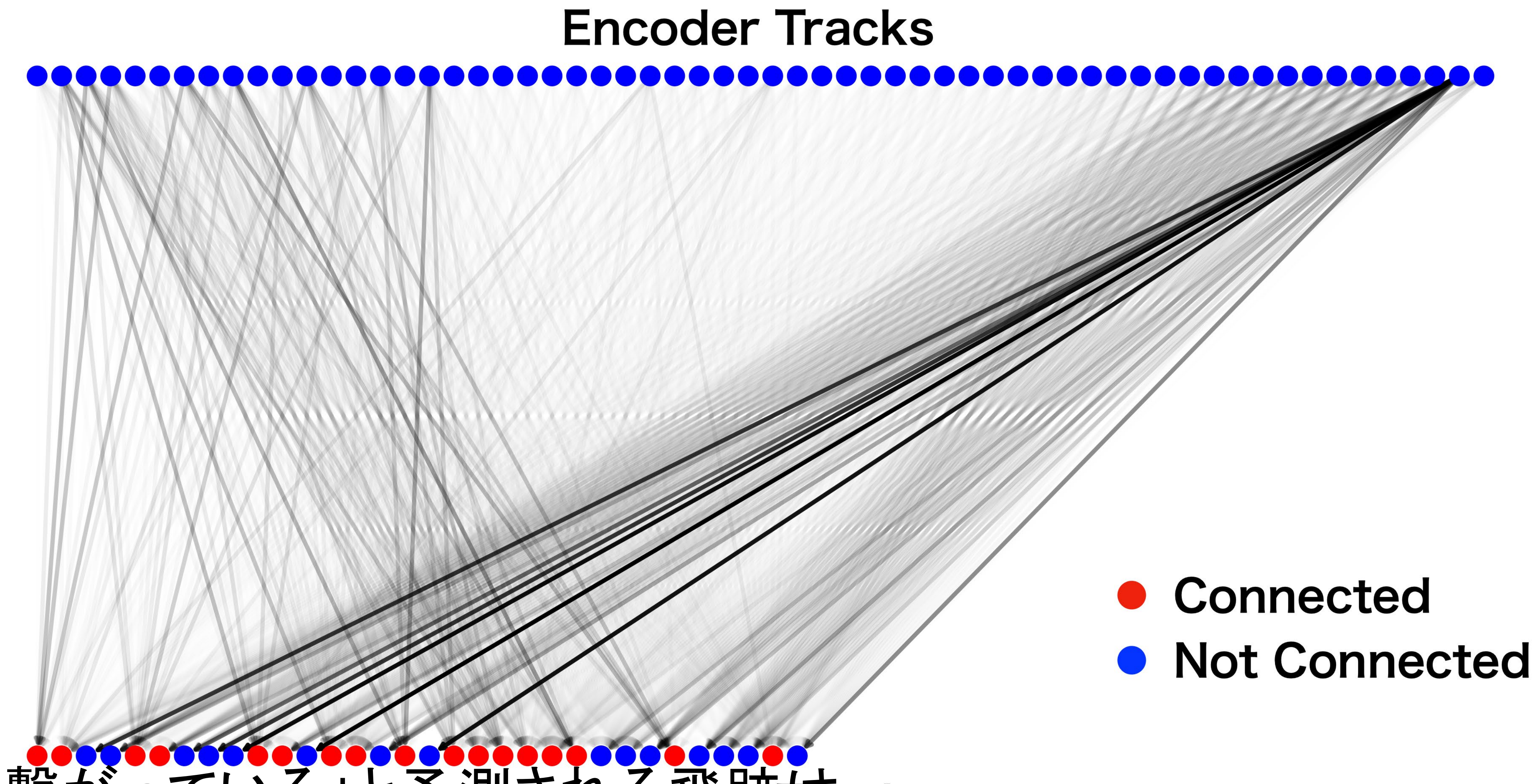


2. 崩壊点検出の為のニューラルネットワーク

Attention

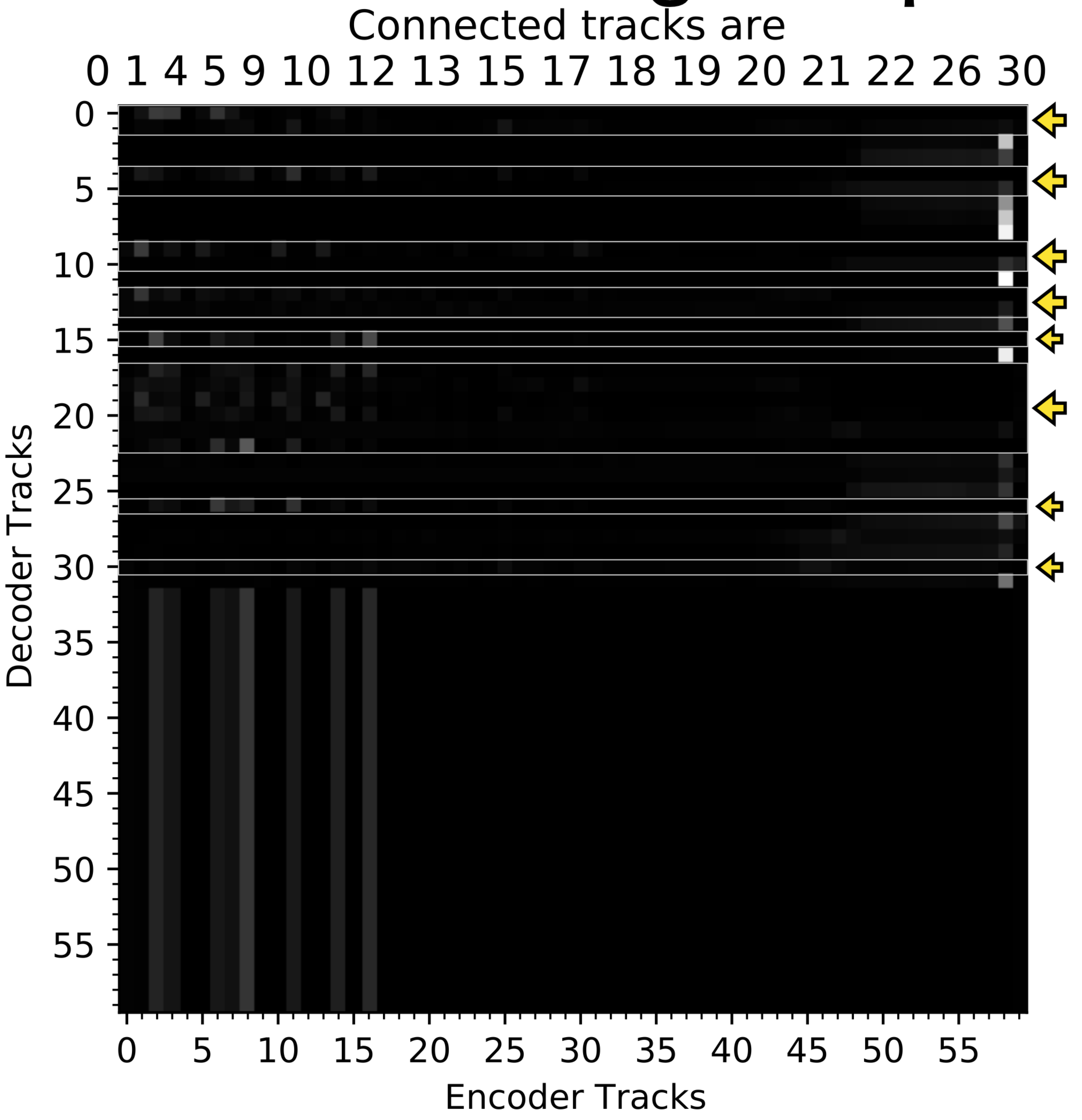
- 各飛跡が事象内の全飛跡に「注意 (Attention)」して欲しい
  「どの飛跡」が「どの飛跡」に注意しているか

Attention Weight Graph



← 線の太さ
色の濃さ ➡□

Attention Weight Map



- 「繋がっている」と予測される飛跡は
  飛跡全体から他の飛跡の情報を受け取れている
- 「どの飛跡」から情報を受け取っているかの調査が必要

3. 深層学習を用いた崩壊点検出

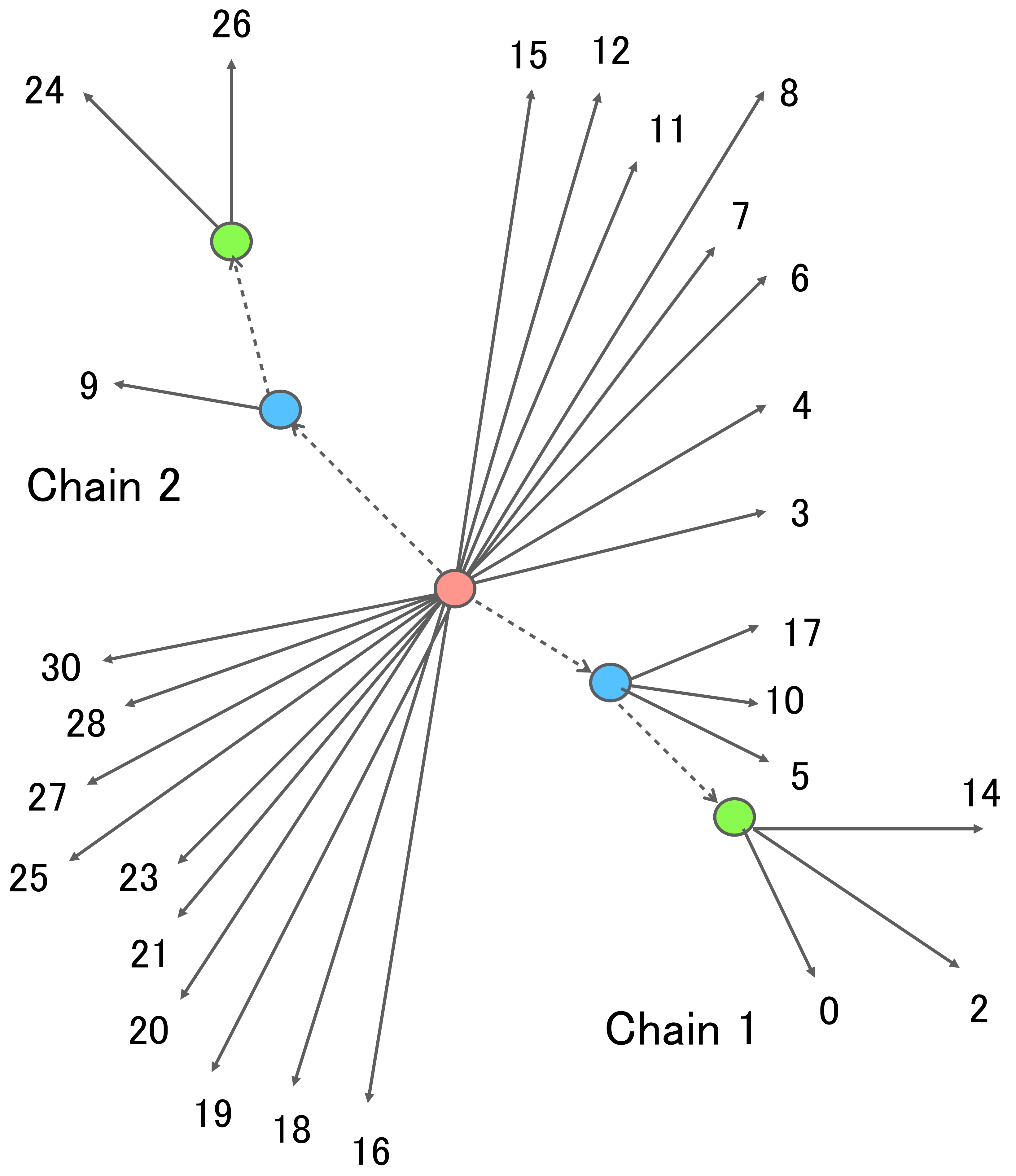
全飛跡 (31 本)

[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 27 28 29 30]

```
True Primary Vertex
[3, 4, 6, 7, 8, 11, 12, 15, 16, 18, 19, 20, 21, 23, 25, 27, 28, 30]
Predict Primary Vertex
[3, 4, 6, 7, 8, 11, 12, 15, 16, 18, 19, 20, 21, 23, 25, 27, 28, 29, 30]
True Secondary Vertex Chain 1
cc : [0, 2, 14]
bb : [5, 10, 17]
one track : []
True Secondary Vertex Chain 2
cc : [24, 26]
bb : []
one track : [9]
Predict Secondary Vertex 0
[24, 26]
Predict Secondary Vertex 1
[2, 10]
Predict Secondary Vertex 2
[5, 17]
Predict Secondary Vertex 3
[0, 14]
------------------------------------------------------------------------
MC Primary / Reco SV : 0.0
MC Others  / Reco SV : 0.0
MC Bottom  / Reco SV : 1.0 Same Chain : 1.0 Same Particle : 0.6666666666666666
MC Charm   / Reco SV : 1.0 Same Chain : 1.0 Same Particle : 0.8
------------------------------------------------------------------------
```

True Others

[1, 13, 22, 29]



3. 深層学習を用いた崩壊点検出

現行の手法 (LCFIPlus) との比較 – フレーバー識別