DEVIEW
2019

엄~청 큰 언어 모델 공장 가동기!

(LaRva: Language Representation by Clova)

이동준, 김성동

Clova AI

NAVER

CONTENTS

DEVIEW
2019

1.Large-Scale PLM Overview

엄~청 큰 언어모델?

리더보드를 잠식하고있는 모델들이 있었으니..

SQuAD2.0

2 Sep 16, 2019	ALBERT (single model) *Google Research & TTIC* https://arxiv.org/abs/1909.11942	88.107	90.902
6 Jul 20, 2019	RoBERTa (single model) *Facebook AI*	86.820	89.795
9 May 21, 2019	XLNet (single model) *Google Brain & CMU*	86.346	89.133
10 Jul 22, 2019	SpanBERT (single model) *FAIR & UW*	85.748	88.709

KorQuAD 1.0

1	2019.06.26	LaRva-Kor-Large+ + CLaF (single) Clova AI LaRva Team	**86.84**	**94.75**
2	2019.06.04	BERT-CLKT-MIDDLE (single model) Anonymous	86.71	94.55
5	2019.07.17	KorBERT Anonymous	86.12	94.02
10	2019.09.20	ETRI BERT (single model) deepfine	84.56	92.91

HotpotQA

3 Oct 1, 2019	EPS + BERT(wwm) (single model) *Anonymous*	
7 Jun 13, 2019	P-BERT (single model) *Anonymous*	
8 Sep 16, 2019	LQR-net 2 + BERT-Base (single model) *Anonymous*	

CoQA

2 Sep 05, 2019	RoBERTa + AT + KD (single model) *Zhuiyi Technology* https://arxiv.org/abs/1909.10772	90.9	**89.2**	90.4
4 Sep 13, 2019	XLNet + Augmentation (single model) *Xiaoming*	89.9	86.9	89.0
6 Mar 28, 2019	ConvBERT (single model) *Joint Laboratory of HIT and iFLYTEK Research*	87.7	84.6	86.8
8 Aug 26, 2019	BERT + AttentionFusionNet (single model) *Beijing Kingsoft AI Lab*	85.4	77.3	83.0

Natural Questions

Rank	Model
1	BERT-dm_v2-ensemble
2	bert-cs
3	bert-dm

SQuAD 2.0: https://rajpurkar.github.io/SQuAD-explorer/,
KorQuAD 1.0: https://korquad.github.io/category/1.0_KOR.html,
CoQA: https://stanfordnlp.github.io/coqa/
HotpotQA: https://hotpotqa.github.io/,
Natural Question: https://ai.google.com/research/NaturalQuestions

리더보드를 잠식하고있는 모델들이 있었으니..

SQuAD2.0

2 Sep 16, 2019	ALBERT (single model) *Google Research & TTIC* https://arxiv.org/abs/1909.11942	88.107	90.902
6 Jul 20, 2019	RoBERTa (single model) *Facebook AI*		
9 May 21, 2019	XLNet (single model) *Google Brain & CMU*		
10 Jul 22, 2019	SpanBERT (single model) *FAIR & UW*		

KorQuAD 1.0

Rank	Reg. Date	Model	EM	F1
-	2018.10.17	**Human Performance**	80.17	91.20
1	2019.06.26	LaRva-Kor-Large+ + CLaF (single) Clova AI LaRva Team	**86.84**	**94.75**

CoQA

2	RoBERTa + AT + KD (single model) *i Technology* *rg/abs/1909.10772*	90.9	**89.2**	90.4
	gmentation (single model) *Xiaoming*	89.9	86.9	89.0
	T (single model) *y of HIT and iFLYTEK esearch*	87.7	84.6	86.8
	ionFusionNet (single model) *Tingsoft AI Lab*	85.4	77.3	83.0

HotpotQA

3 Oct 1, 2019	EPS + BERT(wwm) (single model) *Anonymous*	
7 Jun 13, 2019	P-BERT (single model) *Anonymous*	
8 Sep 16, 2019	LQR-net 2 + BERT-Base (single model) *Anonymous*	

Natural Questions

Rank	Model
1	BERT-dm_v2-ensemble
2	bert-cs
3	bert-dm

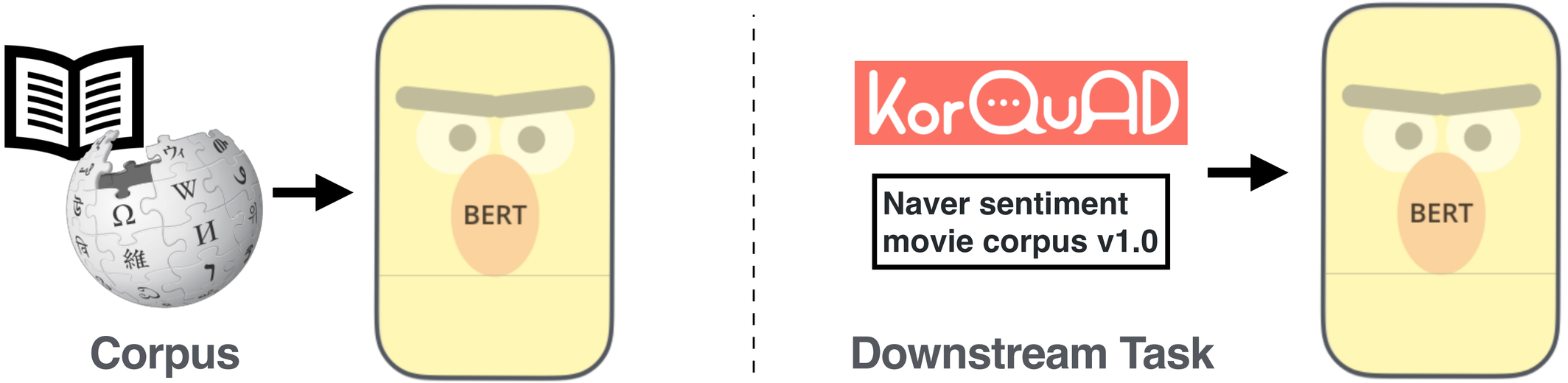
SQuAD 2.0: https://rajpurkar.github.io/SQuAD-explorer/,
KorQuAD 1.0: https://korquad.github.io/category/1.0_KOR.html,
CoQA: https://stanfordnlp.github.io/coqa/
HotpotQA: https://hotpotqa.github.io/,
Natural Question: https://ai.google.com/research/NaturalQuestions

Pre-trained Language Model, NLP의 새로운 패러다임..!

PLM이란?

Pre-training then fine-tuning.

대량의 코퍼스를 Pre-training한 다음, 각 Task에 따라서 fine-tuning 을 진행하는 모델

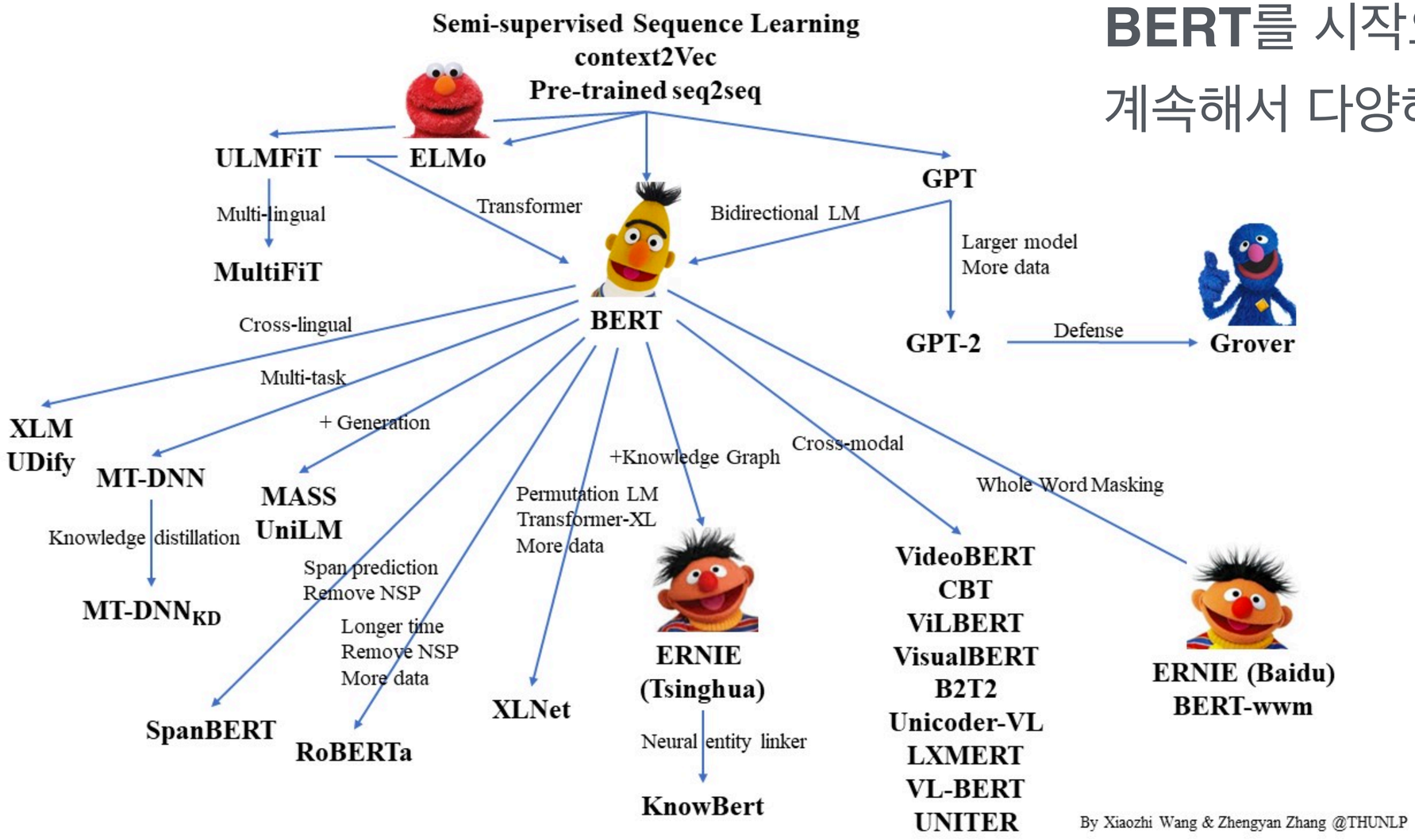


1. Pre-training
(Unsupervised Learning)

2. Fine-tuning
(Supervised Learning)

참고: http://jalammar.github.io/illustrated-bert/

PLM Overview



BERT를 시작으로

계속해서 다양해지는 모델.

출처: https://github.com/thunlp/PLMpapers

PLM Overview

DEVIEW 2019

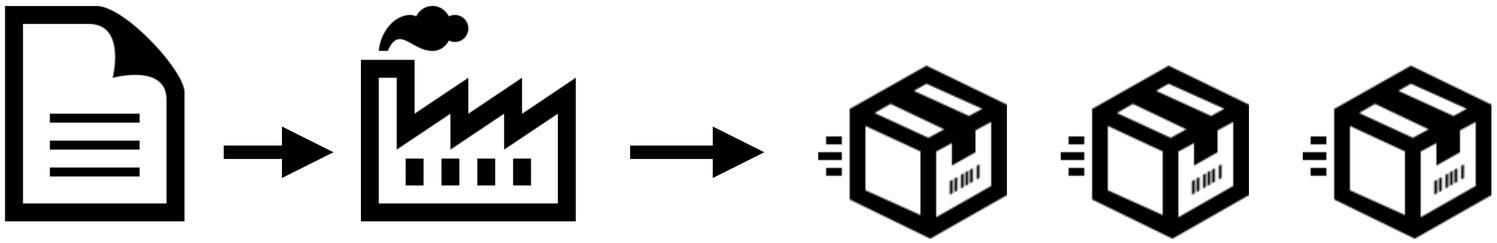
출처: https://medium.com/huggingface/distilbert-8cf3380435b5

오늘 들려드릴 이야기

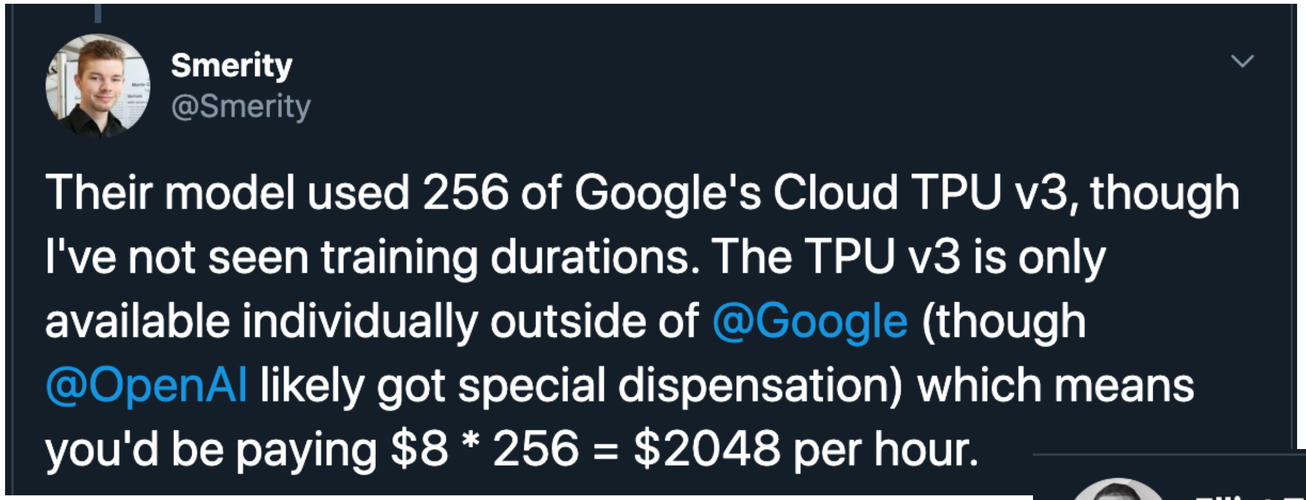
LaRva: **La**nuage **R**epresentation by Clo**va**

이러한 트랜드에 맞춰 한국어, 일본어 등

다양한 언어의 PLM들을 더 빠르고, 더 좋게 만드는 과정.



그런데.. 모델.... 하나에 1억이라고?



GPT-2

시간당 약 250만원씩..
하루에 6천만원..!?

XLNet

모델하나에
2억 9천만원 !?

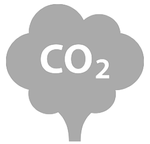
출처: GPT-2: https://twitter.com/Smerity/status/1096189352743301120

XLNet: https://twitter.com/eturner303/status/1143174828804857856

논문 기준, 학습에 필요한 가격

* TPU는 1 device -> 4 chips -> 8 cores 입니다.

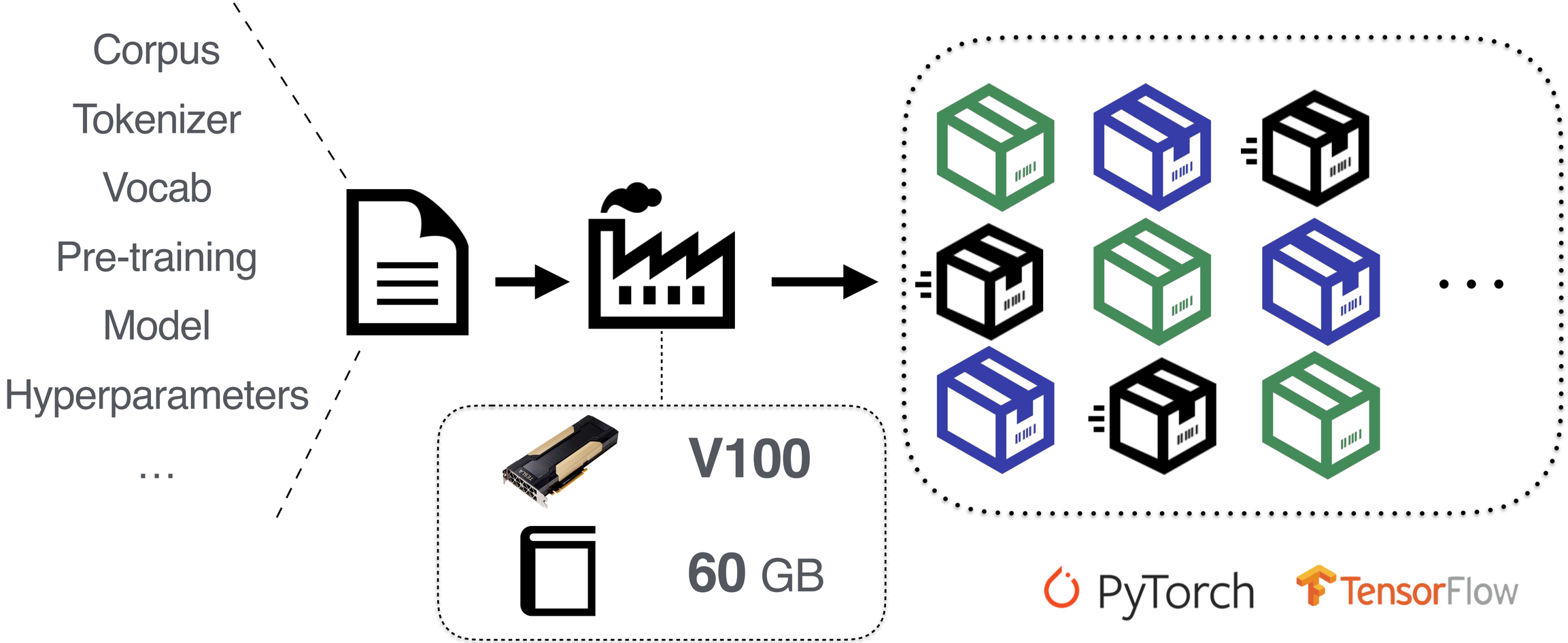
Model	Size	TPU ($ per hour)	TPU Count (device)	Training Time	Cost (USD)	CO2 emissions (lbs)
BERT	24 Layers (340M)	v2 ($4.5)	16	4 days	$6,912 (약 850만원)	1428
GPT-2	48 Layers (1542M)	v3 ($8)	32	7 days	$43,008 (약 5,100만원)	2516
XLNet	24 Layers (365M)	v3 ($8)	128	2.5 days	$61,440 (약 7,300만원)	-



* NY ↔ SF Air Travel: 1924 (lbs)

참고: https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/,
Energy and Policy Considerations for Deep Learning in NLP, 2019 E Strubell

우리에게 주어진 자원과 요구사항들



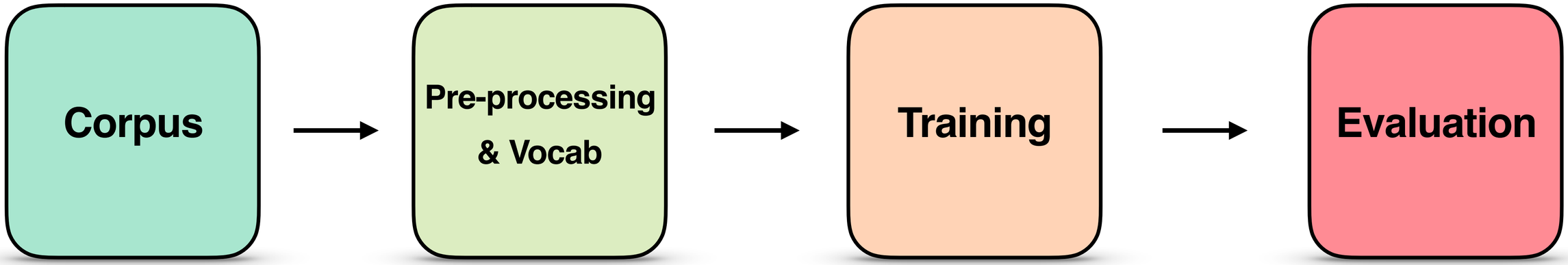
비싼 만큼,
일의 효율이 중요하다!

2. LaRva
공장 최적화

LaRva Pipeline v0.1.0

공개코드를 기반으로 시작!

(https://github.com/google-research/bert)



60G

+ 600G (TFRecords)

약 10일 소요

학습이 끝난 후,
하나하나 직접 평가

Official code의 문제점

- 비싼 TPU, 실험은 기본적으로 시행착오
  - 한번 시도에 돈이 **1천만원**이 날아간다면!?
- 학습이 전부 끝난 후에야 평가
  - 1천만원과 더불어서 **4일**이라는 시간까지!?
- Pre-training 에 사용되는 데이터(TFRecords) 생성에 필요한 저장공간
  - 예) Corpus 17GB —> 1개 실험에 **170 GB** (학습용 데이터 10벌 미리 생성)

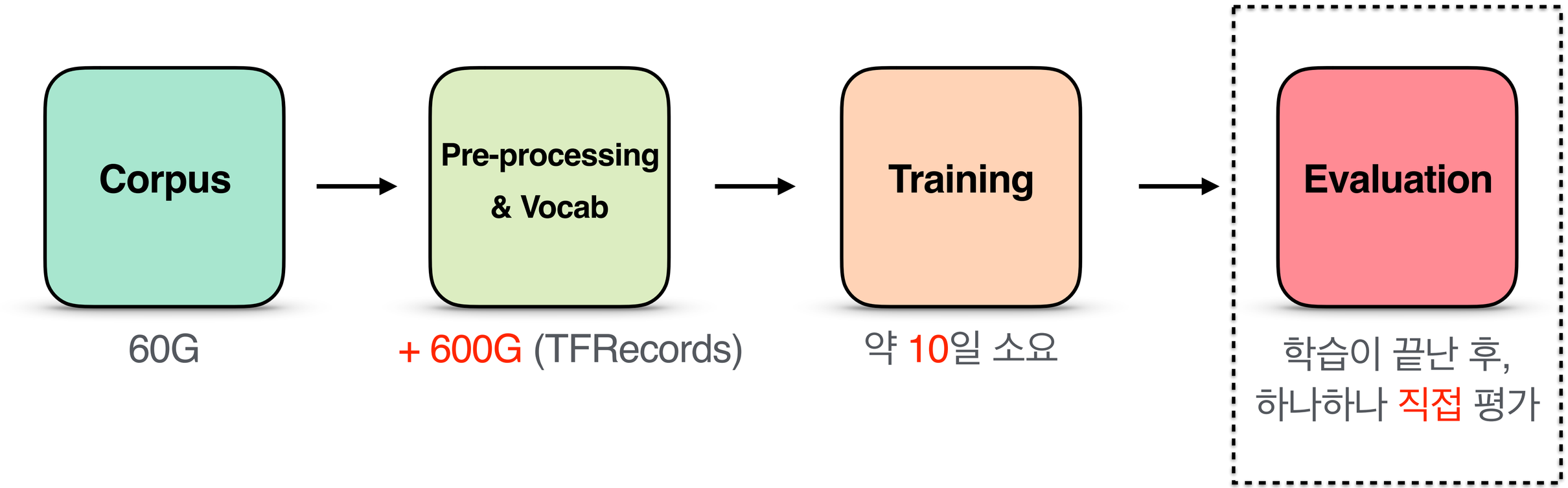
BERT-Large

BERT 이미지: http://jalammar.github.io/illustrated-bert/

2-1. 평가를 손쉽게

(BERT Base 기준)

LaRva Pipeline v0.1.0



평가에서의 요구사항들

시행착오를 빠르게 발견하자!

- 총 1M Steps 훈련 기준, 매 100K Steps (약 1일) 마다 체크포인트를 받아, Downstream Task로 검증하자.
- 총 10개의 체크포인트, 준비된 평가용 Downstream Task는 8개.
  - 그렇다면.. 하나의 모델을 학습할 때, **80번**이나 평가를 돌려야 한다..!

결과가 한 눈에 들어오도록 하자!

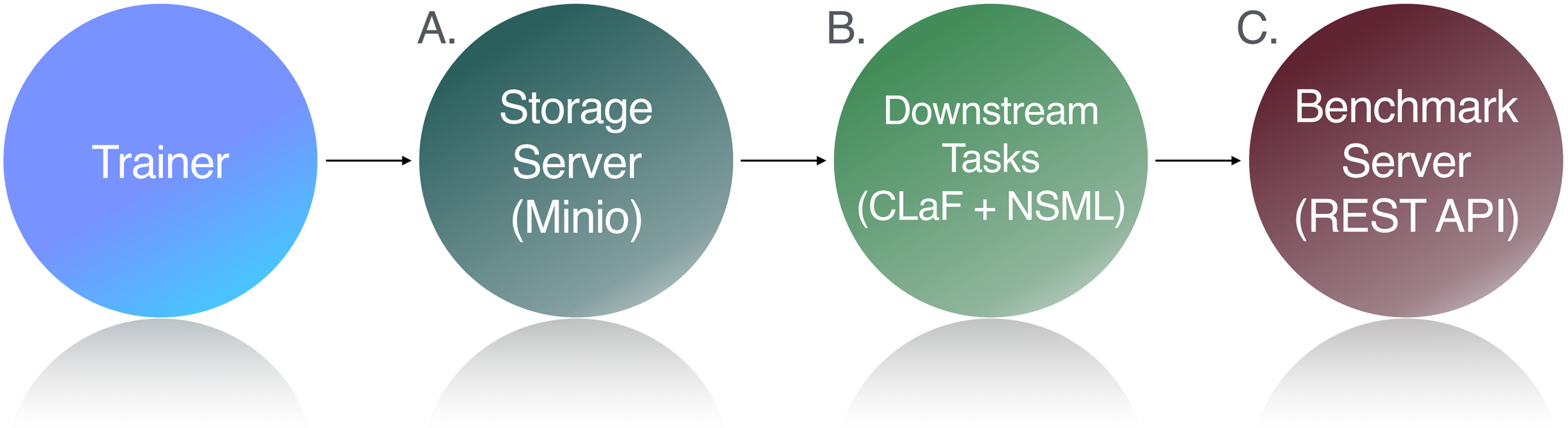
- Step 별로 모델 성능이 어떻게 나아지는지 알고 싶어요.
- 옵션 별로 성능차이를 보고 싶어요.

평가와 성능 리포트를 자동화 & 시각화!

A. Fine-tuning를 위해서 **체크포인트를 한 곳에 모으기.**

B. 다양한 Downstream Task를 돌릴 수 있도록 **실험의 옵션 미리 정의**
   모델과 더불어 Tokenizer 등, 필요한 옵션 **간단하게 변경 가능**

C. 평가 결과를 저장할 수 있는 **REST API 서버 + Dashboard 페이지**



Minio: Object Storage for AI

AWS S3와 같은 인터페이스 제공

Checkpoint upload:

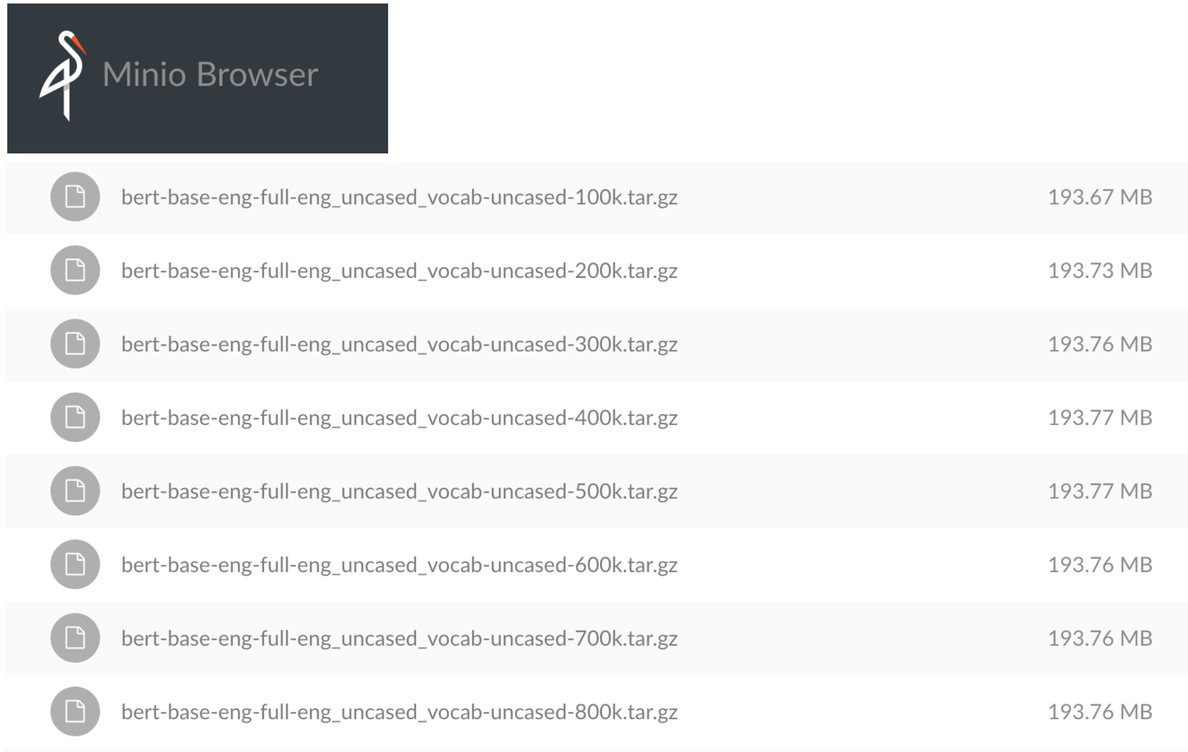
config와 weight를 복사해서 압축 후,

서버에 업로드

Download & Load:

Bucket policy를 통해서, URL 접근이 가능

URL만 입력하면 다운로드 후, 모델 로드



Minio: https://github.com/minio/minio

Downstream Tasks

BERT 논문에서 모델의 성능 평가용으로 사용된 데이터셋 (e.g. GLUE)

- 공개된 데이터셋과 내부용 데이터셋을 모아 **KLUE/JLUE 데이터셋** 확보

* **KLUE / JLUE**: 한국어, 일본어 언어이해 벤치마크

Task 종류

기계 독해	감정 분석	유사한 문장	다음 문장 예측
의도파악	개체명 인식	의미역 결정	

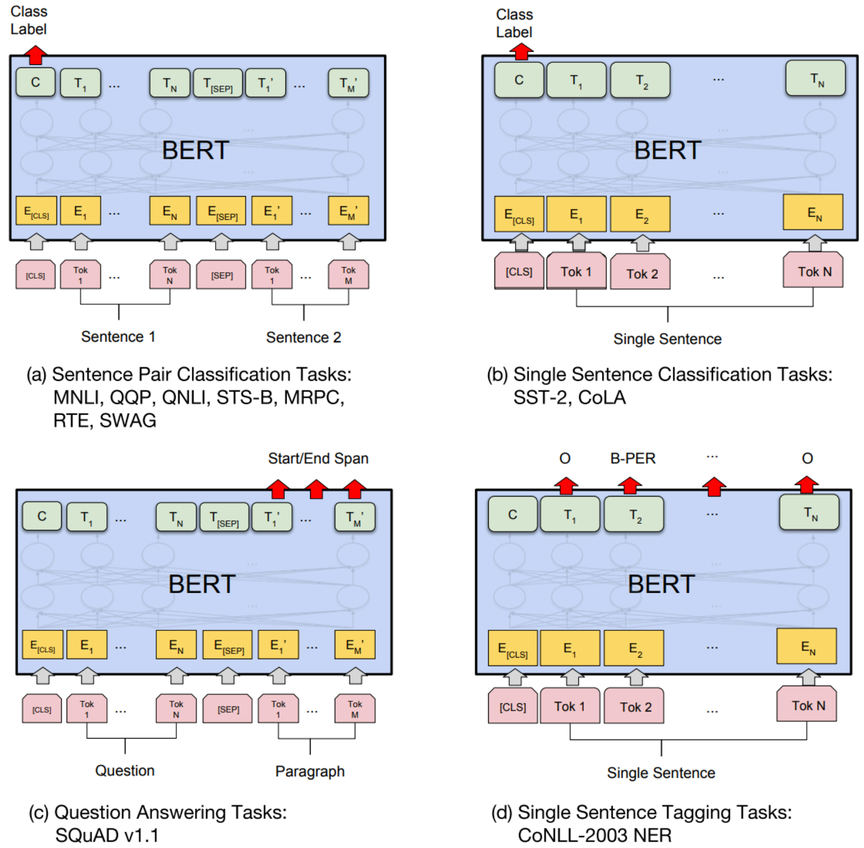


Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

이미지: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding - 2018 J Devlin

Fine-tuning 에 필요한 요구사항들

Reproducible, 재현이 가능해야 한다.

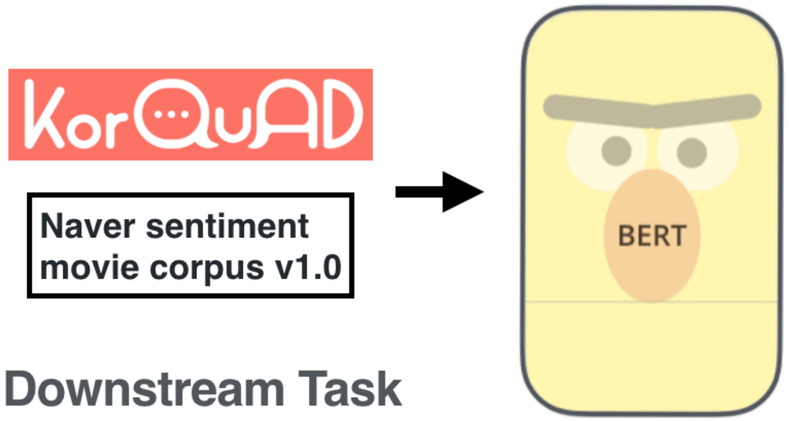
각 모델의 **Hyperparameter 들은 고정**이 되어 한다.

- 각 논문의 고정된 값들을 사용

다양한 **Task들을 쉽게 추가**해서, 돌릴 수 있어야 한다.

- 한국어/일본어/영어 등 **다양한 언어 지원**도 필요
- Tokenizer 등 다양한 **옵션에 대해 변경이 용이**해야 한다.

자동화를 위해서 실험에 대한 준비가 전부 되어있어야 한다.

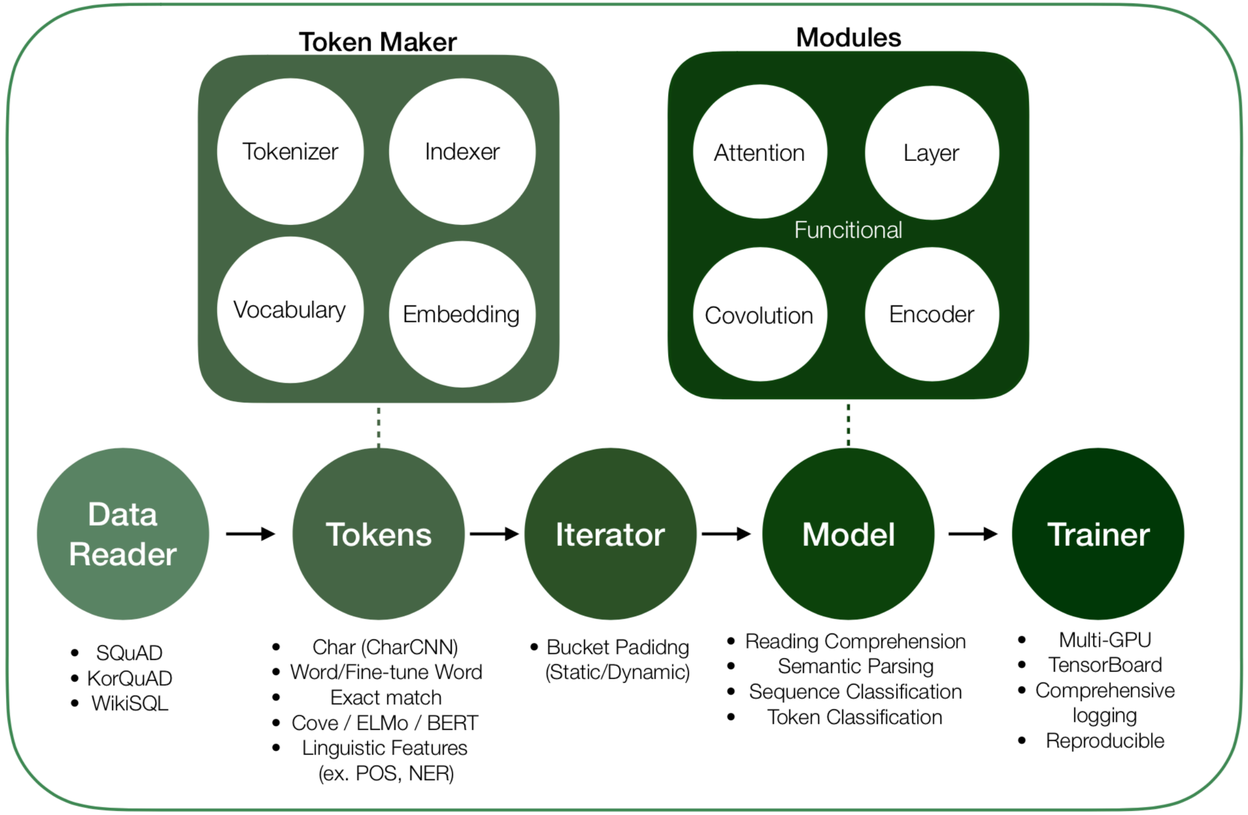


2. Fine-tuning
(Supervised Learning)

출처: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

CLaF: Clova Language Framework

- **Open-Source** 자연어처리 프레임워크
  (PyTorch 기반)
- **추상화와 모듈**
  - **Multi-lingual 지원** (GLUE, KLUE, and JLUE)
  - 다양한 Task 쉽게 추가, Option 변경 (변경의 용의성)
- **Reproducible** + Logging
- **NSML** (네이버 딥러닝 클라우드 플랫폼) 호환
- **BaseConfig** 미리 실험 정의
  (Hyperparameters 고정)



CLaF: https://github.com/naver/claf

BaseConfig: 미리 실험들을 정의

- 각 컴포넌트의 Parameter에 연결하여 실험을 진행
- 정의 되어 있는 Config 값을 사용
  - KLUE/JLUE 미리 정의
- Argument를 통해서 변경 가능

=> 실험에 대한 셋팅과 Hyperparameter가 고정되었으므로, 필요한 옵션만 바꿔서 실험 진행!

```
{
    "data_reader": {
        "dataset": "squad_bert",
        "train_file_path": "korquad/train.json",
        "valid_file_path": "korquad/dev.json",
        "squad_bert": {
            "lang_code": "ko",
            "max_seq_length": 512,
            "context_stride": 128,
            "max_question_length": 64
        }
    },
    "iterator": {
        "batch_size": 12
    },
    "token": {
        "names": ["feature"],
        "types": ["feature"],
        "tokenizer": {
            "subword": {
                "name": "wordpiece",
                "wordpiece": {
                    "vocab_path": "kor_32k_v1_vocab.txt"
                }
            },
            "word": {
                "name": "bert_basic",
                "bert_basic": {
                    "do_lower_case": false
                }
            }
        },
        "feature": {
            "vocab": {
                "pretrained_path": "kor_32k_v1_vocab.txt",
                "pretrained_token": "all"
            },
            "indexer": {
                "do_tokenize": false
            }
        }
    },
    "model": {
        "name": "bert_for_qa",
        "bert_for_qa": {
            "pretrained_model_name": "larva-kor-small-cased",
            "answer_maxlen": 30
        }
    },
    "trainer": {
        "log_dir": "logs/klue/korquad",
        "num_epochs": 5,
        "early_stopping_threshold": 10,
        "metric_key": "f1",
        "verbose_step_count": 100,
        "eval_and_save_step_count": 1000
    },
    "optimizer": {
        "learning_rate": 3e-5,
        "op_type": "adamw",
        "adamw": {
            "weight_decay": 0.01
        },
        "lr_scheduler_type": "warmup_linear",
        "warmup_linear": {
            "warmup_proportion": 0.1
        }
    },
    "seed_num": 42
}
```

KorQuAD BaseConfig 예시

Fine-tuning 실행 예시


```
claf git:(master) ✗ python train.py —base_config klue/korquad_bert —bert_for_qa.pretrained_model_name “modelA-100k”
```

```
claf git:(master) ✗ python train.py —base_config klue/korquad_bert —bert_for_qa.pretrained_model_name “modelA-200k”
```

KorQuAD Task에 model1의 100k Step, 200k Step Fine-Tuning 진행.

Fine-tune 결과:

```
Metric Logs.
{
    "best_epoch": 3,
    "best_global_step": 14000,
    "best": {
        "valid/loss": 0.5234435290928358,
        "valid/epoch_time": 60.746843099594116,
        "valid/em": 84.67267059231035,
        "valid/f1": 92.95460178922072
    },
    "best_score": 92.95460178922072,
```

```
Metric Logs.
{
    "best_epoch": 5,
    "best_global_step": 25000,
    "best": {
        "valid/loss": 0.6766484450896716,
        "valid/epoch_time": 64.26702761650085,
        "valid/em": 84.84586075510911,
        "valid/f1": 93.33498288786473
    },
    "best_score": 93.33498288786473,
```

Benchmark Server: Table

- 각 Task의 성능을 확인할 수 있는 기본적이 Table. (DataTables 기반)
- 논문에서와 마찬가지로 가장 높은 성능에 **Bold** 처리
- Task 별로 정렬, 보고 싶은 모델을 골라서 볼 수 있는 Search 제공

DataTables

GLUE Benchmark

Search:

Model	CoLA (Matt)	SST-2 (Acc)	MRPC (F1+Acc/2)	STS-B (P+S/2)	QQP (F1+Acc/2)	MNLI (Acc)	MNLImm (Acc)	QNLI (Acc)	RTE (Acc)	WNLI (Acc)
bert-large-uncased	**61.440**	**94.037**	89.537	**89.991**	88.745	**86.480**	**86.025**	89.934	70.036	56.338
bert-large-eng-multi_task-5_epoch-uncased	60.241	93.922	**92.119**	88.639	88.337	85.818	85.852	**90.335**	**83.394**	54.930
bert-base-uncased	57.824	93.234	90.265	87.146	88.661	84.483	84.723	88.660	66.065	57.746
bert-base-eng-multi_task-5_epoch-uncased	55.255	93.349	91.519	87.478	87.762	84.157	84.194	88.364	77.617	45.070

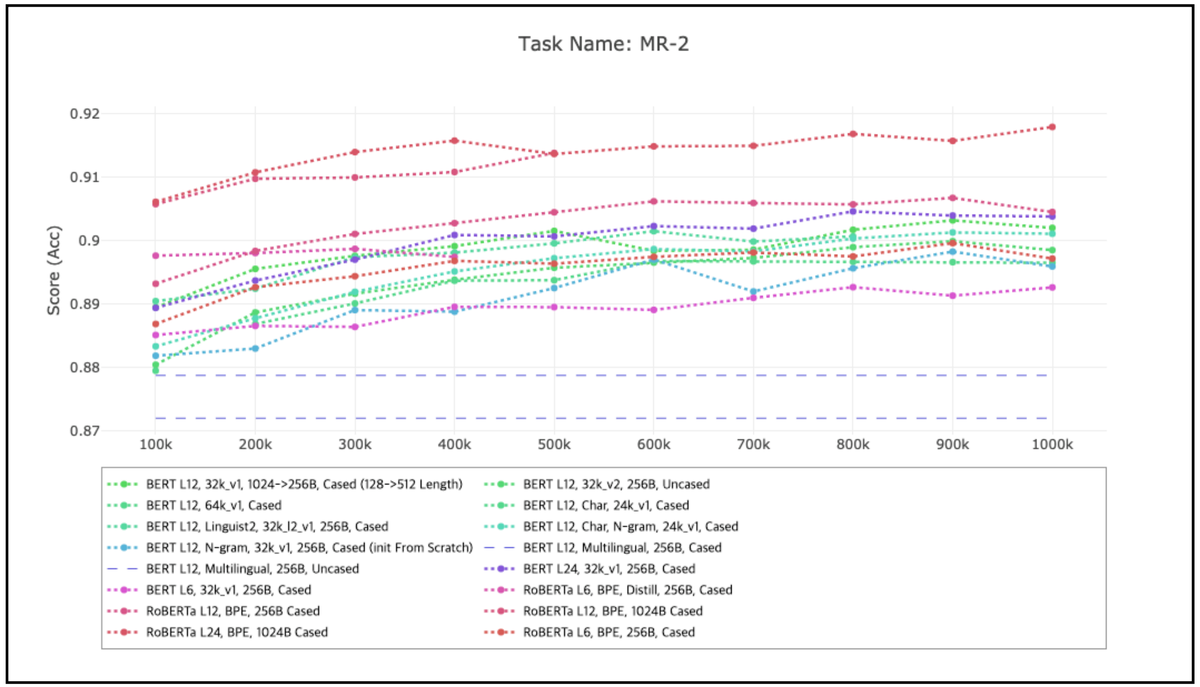
Benchmark Server: Chart

DEVIEW 2019

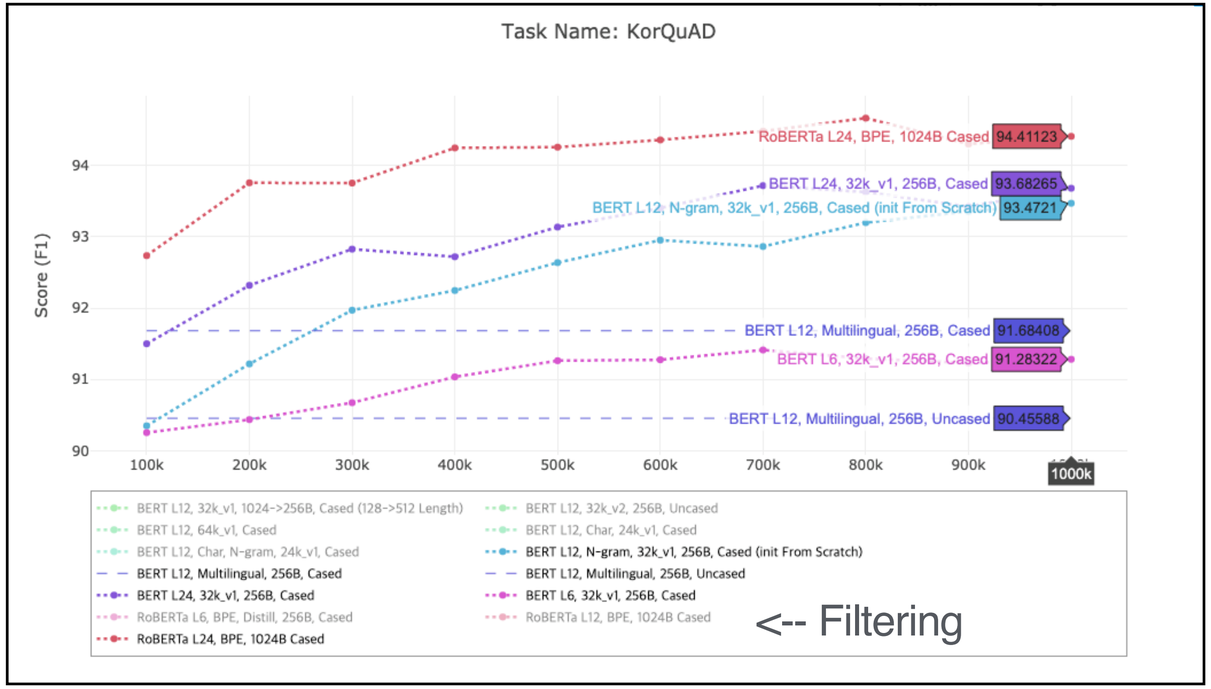
원하는대로 싶은대로 컨트롤이 가능한 Interactive Chart (plotly기반)

plotly

☑ Step 별로 모델 성능이 어떻게 나아지는지 알고 싶어요.

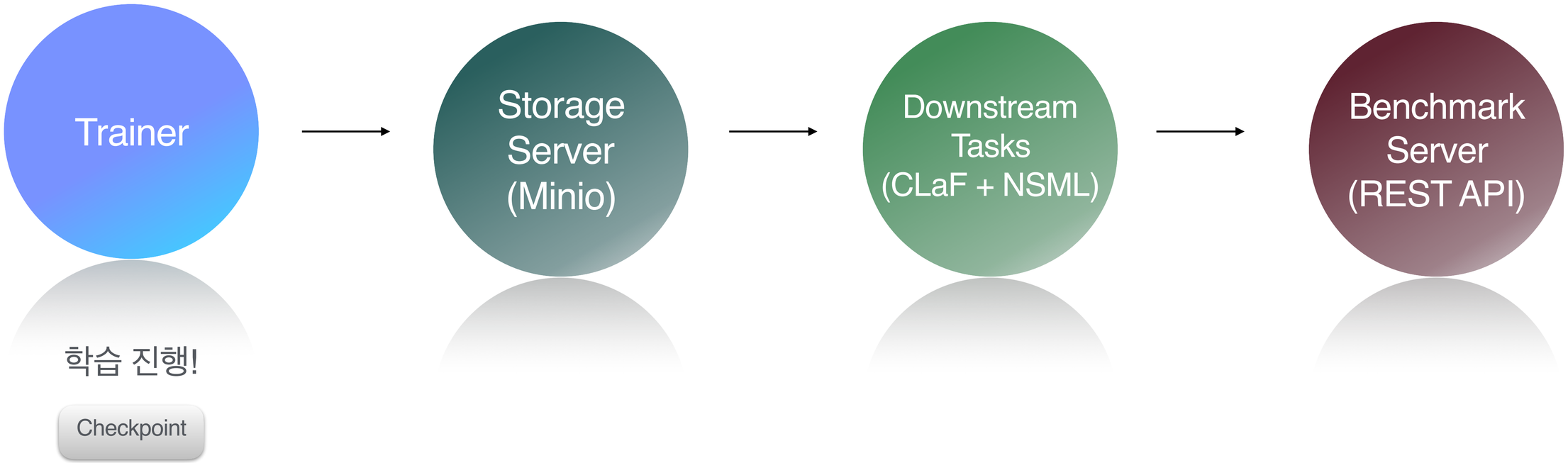


☑ 옵션 별로 성능차이를 보고 싶어요.

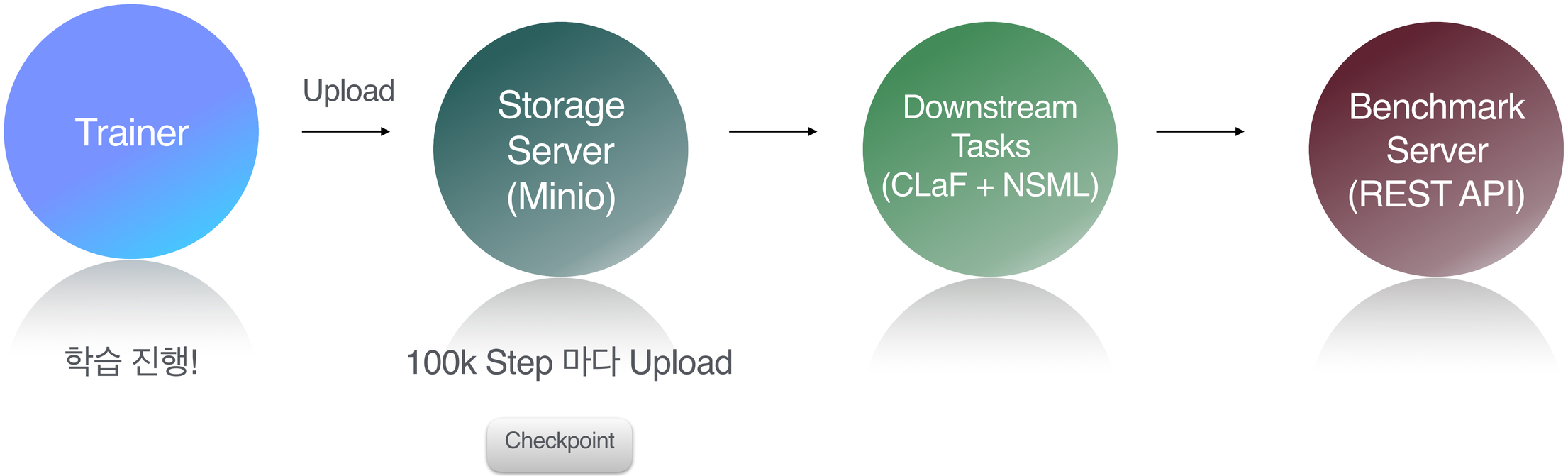


평가와 성능 리포트를 자동화!

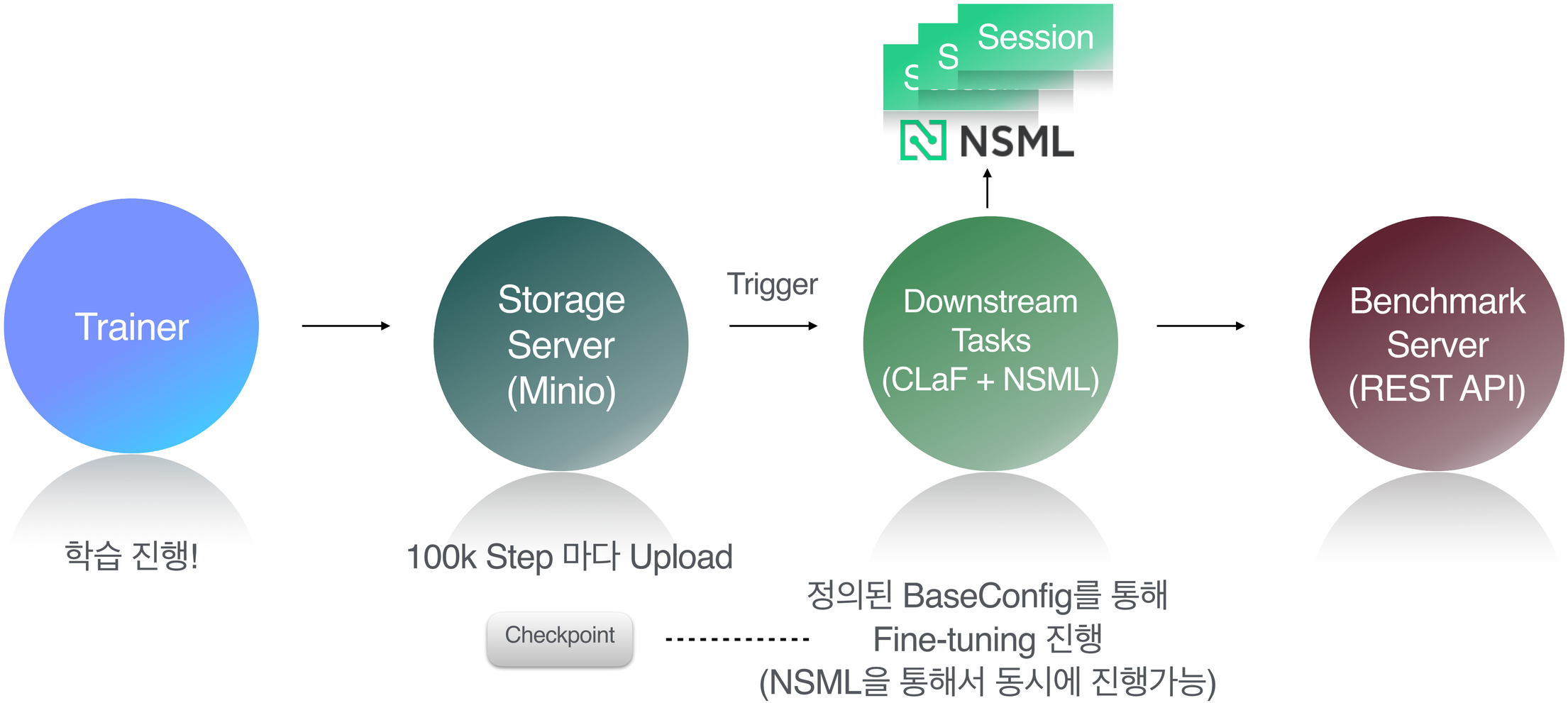
DEVIEW 2019



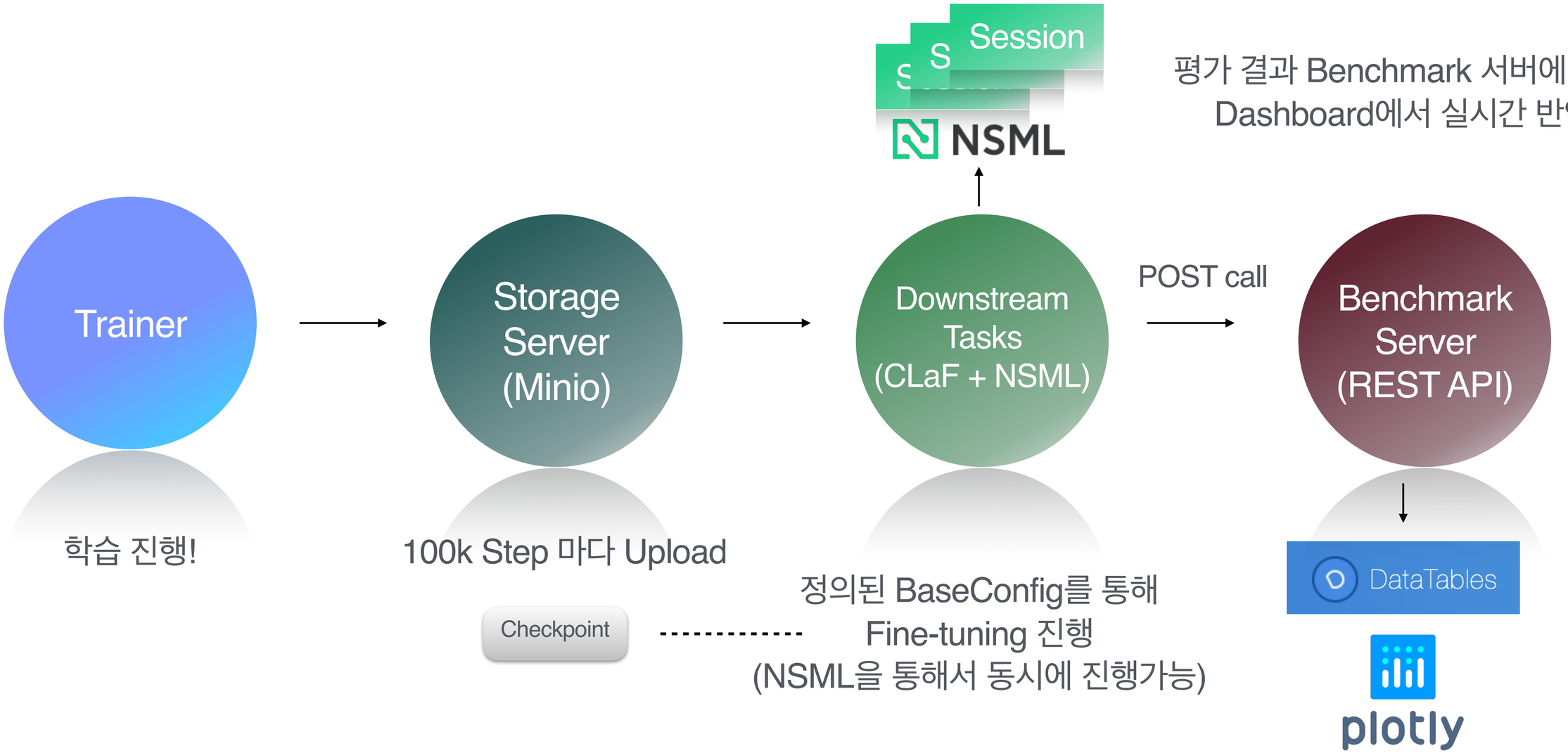
평가와 성능 리포트를 자동화!



평가와 성능 리포트를 자동화!



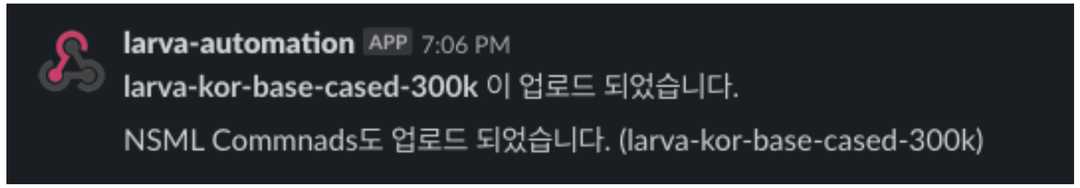
평가와 성능 리포트를 자동화!



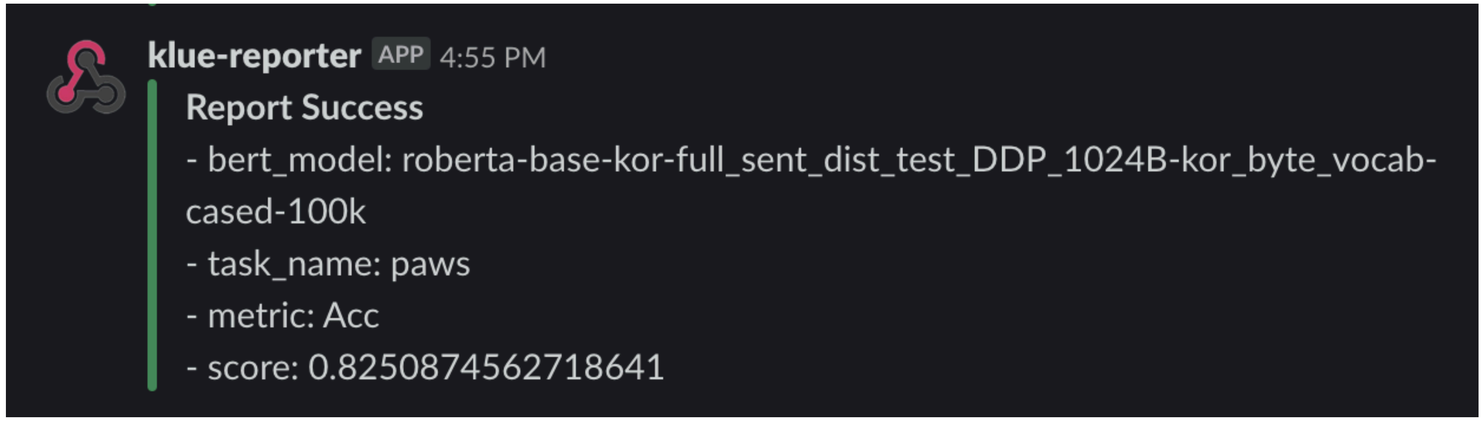
실시간 알림 받기

Slack 연동을 통해서 학습 및 평가의 진행상황을 실시간으로 파악.

1. Minio 서버에 체크포인트 업로드



2. Downstream Task 결과

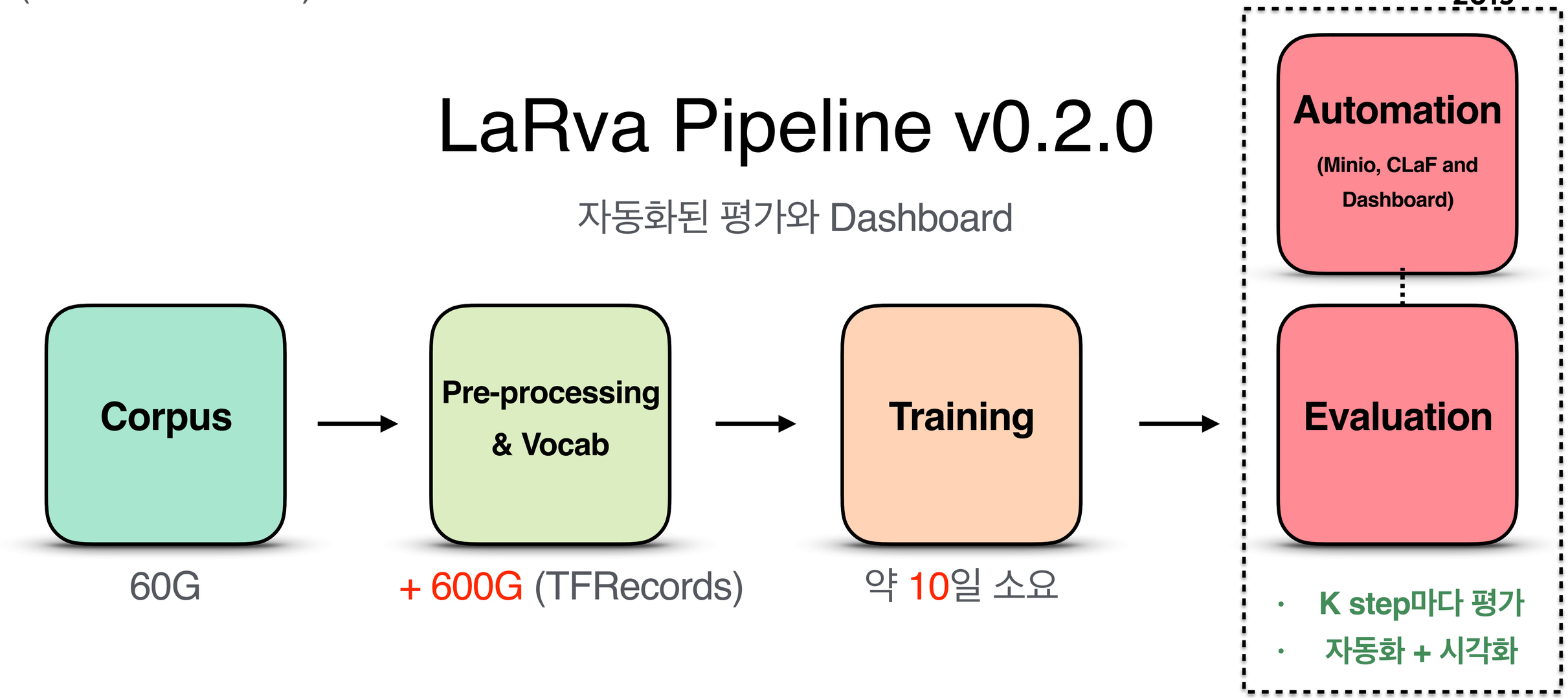


(BERT Base 기준)

DEVIEW 2019

LaRva Pipeline v0.2.0

자동화된 평가와 Dashboard

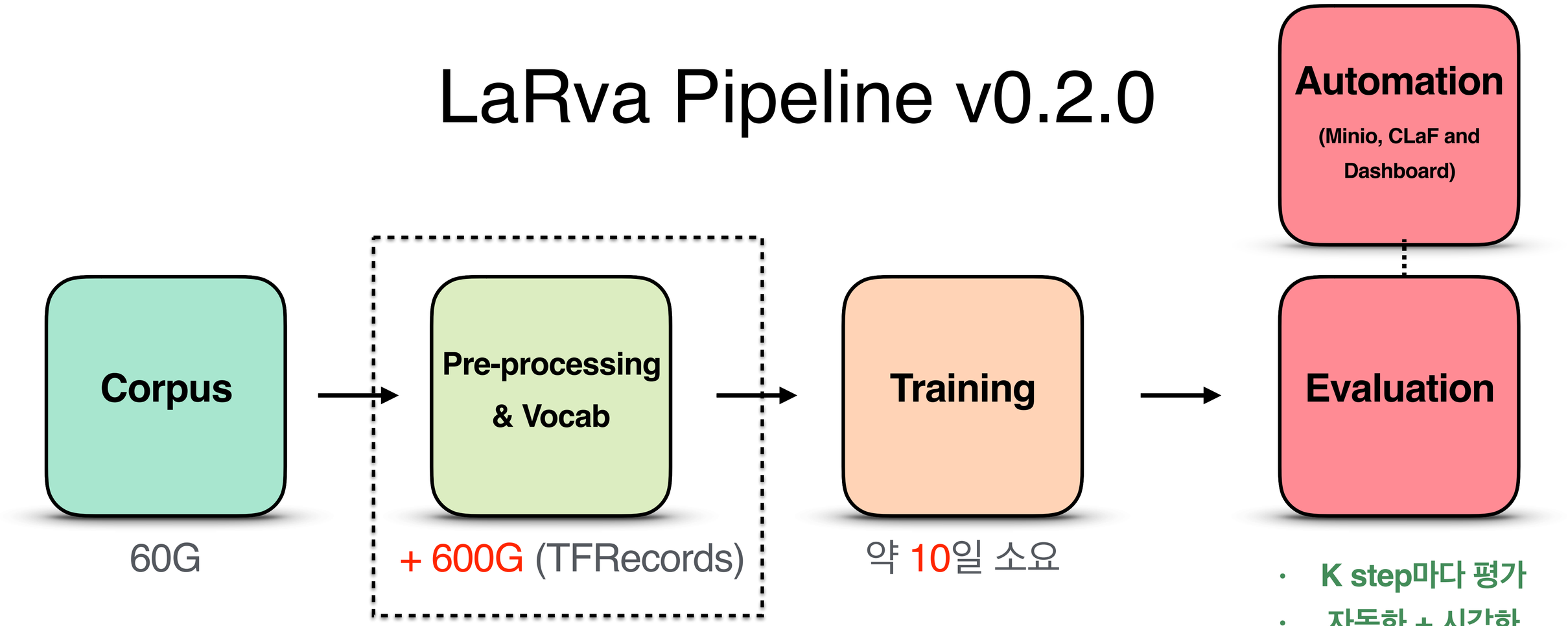


2-2. 전처리를 효율적으로

(BERT Base 기준)

DEVIEW 2019

LaRva Pipeline v0.2.0



Multi-lingual vocab from Google

- `BERT-Base, Multilingual Cased` : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters

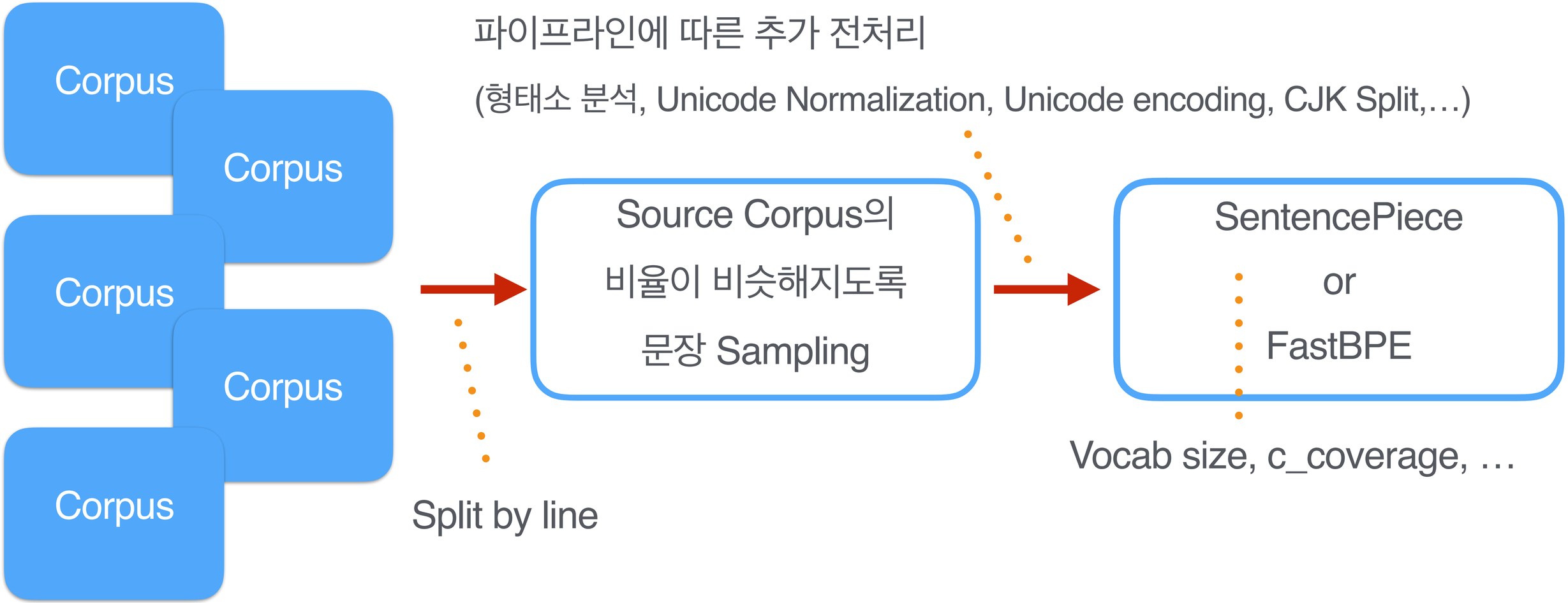
구글에서 공개한 한국어를 포함한 104개 언어로 학습된 BERT-BASE 모델

- Vocab size : **약 10만 (119,547)**
- 한국어 Unicode를 포함한 토큰의 비율 : **2.7 % (3,271)**

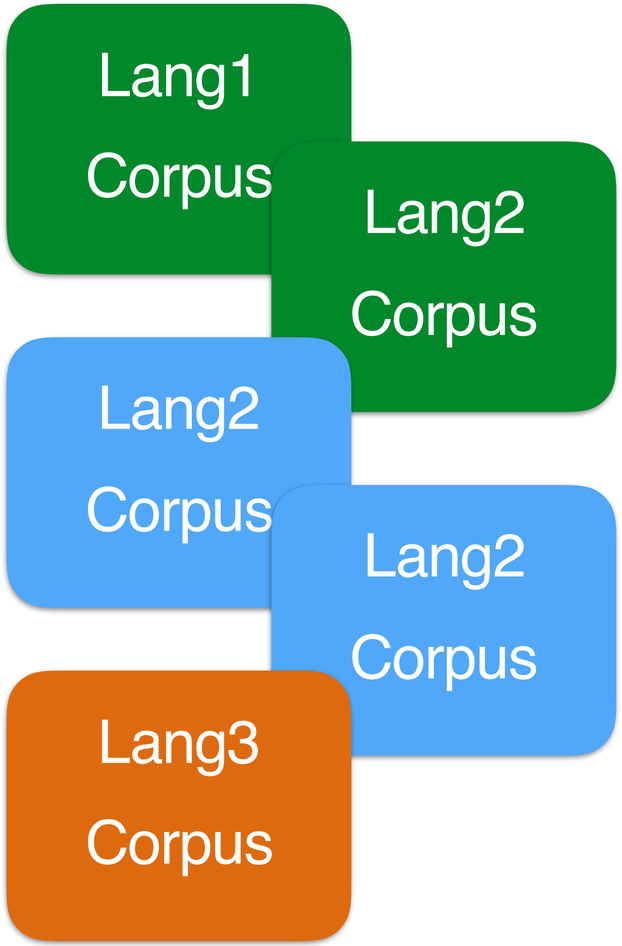
Korquad devset의 Paragraphs 기준	Avg tokenized sequence length	UNK Ratio
Multilingual-cased vocab (Google)	329	6%

https://github.com/google-research/bert

Vocab 구축 파이프라인



Vocab 구축 파이프라인



파이프라인에 따른 추가 전처리

(형태소 분석, Unicode Normalization, Unicode encoding, CJK Split,…)

Source Corpus의
비율이 비슷해지도록
문장 Sampling

SentencePiece
or
FastBPE

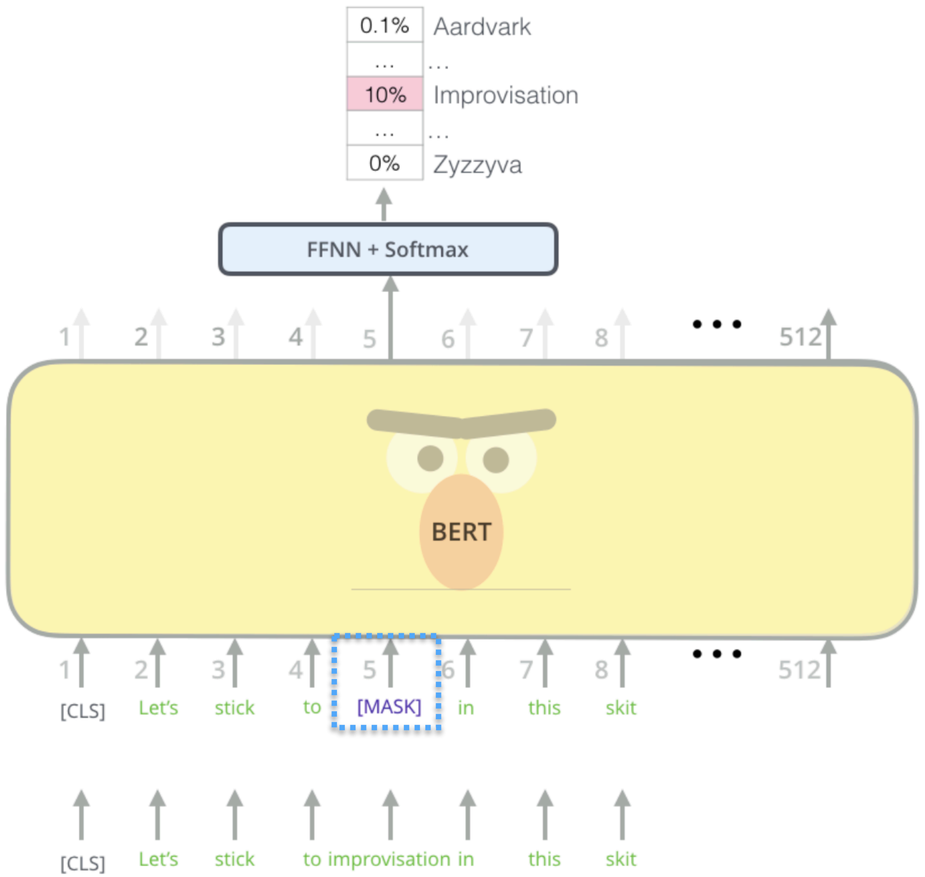
특히 **Multi-lingual vocab**을 만들 때 더 중요!!

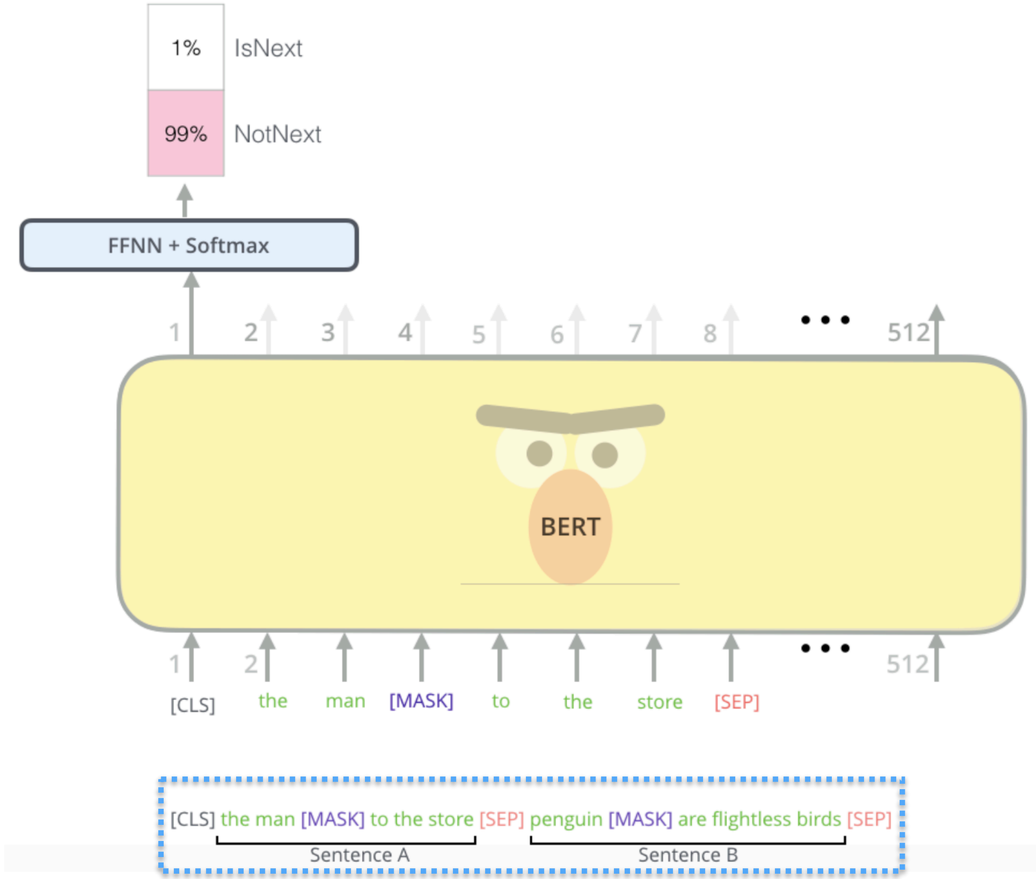
(Mono-lingual인 경우 구어체와 문어체의 비율 고려)

한국어 Vocab Benchmark

	Vocab size	Korean Unicode Ratio	Avg tokenized sequence length	UNK Ratio
Multilingual-cased vocab (Google)	119547	2.7%	329	6%
Korean vocab v1 (cased)	32000	83%	233	4%
Korean vocab v2 (cased)	64000	85%	213	2%
Korean vocab v3 (cased)	96000	86%	203	1.5%

MLM과 NSP





이미지 출처 : http://jalammar.github.io/illustrated-bert/

MLM과 NSP

Input Example

**데뷰에서 라바를 발표 중입니다. MLM과 NSP의 인풋이 어떻게 만들어질까요?
한번 알아보도록 합시다!**

```
tokens: [CLS] 데 ##뷰 ##에서 [MASK] ##바를 발표 중 ##입니다 . ML ##M ##과 [MASK] ##의 인 ##풋 ##이 어떻
게 만들어 ##질 ##까요 ? [SEP] 한번 알아보 ##도록 ##퐁 ##시다 [MASK] [SEP]
segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
is_random_next: False
masked_lm_positions: 4 13 25 27 29
masked_lm_labels: 라 NSP 알아보 합 !
```

```
tokens: [CLS] 데 ##뷰 ##에서 라 ##바를 발표 [MASK] ##입니다 . ML ##M ##과 [MASK] [MASK] 인 ##풋 ##이
어떻게 만들어 ##질 ##까요 ? [SEP] [MASK] 랜덤 텍스트 [SEP]
segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
is_random_next: True
masked_lm_positions: 7 13 14 24
masked_lm_labels: 중 NSP ##의 이것은
```

MLM과 NSP

Input Example

**데뷰에서 라바를 발표 중입니다. MLM과 NSP의 인풋이 어떻게 만들어질까요?
한번 알아보도록 합시다!**

```
tokens: [CLS] 데 ##뷰 ##에서 [MASK] ##바를 발표 중 ##입니다 . ML ##M ##과 [MASK] ##의 인 ##풋 ##이 어떻게 만들어 ##질 ##까요 ? [SEP] 한번 알아보 ##도록 ##퐁 ##시다 [MASK] [SEP]
segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
is_random_next: False
masked_lm_positions: 4 13 25 27 29
masked_lm_labels: 라 NSP 알아보 합 !
```

```
tokens: [CLS] 데 ##뷰 ##에서 라 ##바를 발표 [MASK] ##입니다 . ML ##M ##과 [MASK] [MASK] 인 ##풋 ##이 어떻게 만들어 ##질 ##까요 ? [SEP] [MASK] 랜덤 텍스트 [SEP]
segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
is_random_next: True
masked_lm_positions: 7 13 14 24
masked_lm_labels: 중 NSP ##의 이것은
```

MLM과 NSP

Input Example

**데뷰에서 라바를 발표 중입니다. MLM과 NSP의 인풋이 어떻게 만들어질까요?
한번 알아보도록 합시다!**

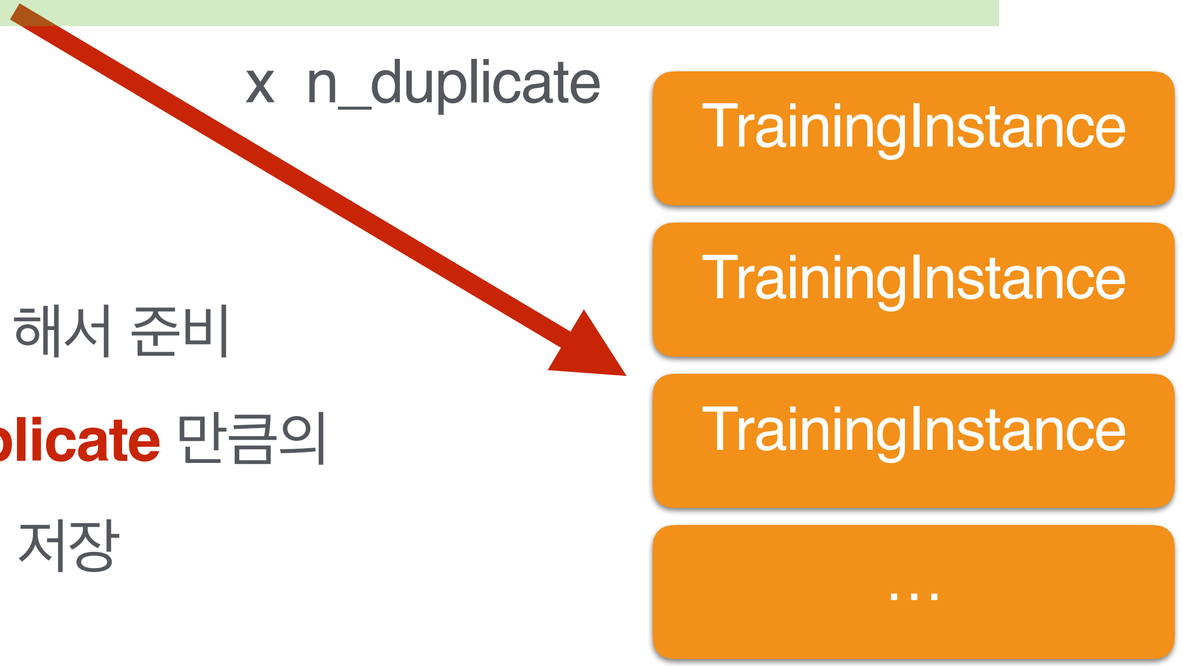
```
tokens: [CLS] 데 ##뷰 ##에서 [MASK] ##바를 발표 중 ##입니다 . ML ##M ##과 [MASK] ##의 인 ##풋 ##이 어떻
게 만들어 ##질 ##까요 ? [SEP] 한번 알아보 ##도록 ##퐁 ##시다 [MASK] [SEP]
segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
is_random_next: False
masked_lm_positions: 4 13 25 27 29
masked_lm_labels: 라 NSP 알아보 합 !
```

```
tokens: [CLS] 데 ##뷰 ##에서 라 ##바를 발표 [MASK] ##입니다 . ML ##M ##과 [MASK] [MASK] 인 ##풋 ##이
어떻게 만들어 ##질 ##까요 ? [SEP] [MASK] 랜덤 텍스트 [SEP]
segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
is_random_next: True
masked_lm_positions: 7 13 14 24
masked_lm_labels: 중 NSP ##의 이것은
```

TFRecord

Input Example

**데뷰에서 라바를 발표 중입니다. MLM과 NSP의 인풋이 어떻게 만들어질까요?
한번 알아보도록 합시다!**

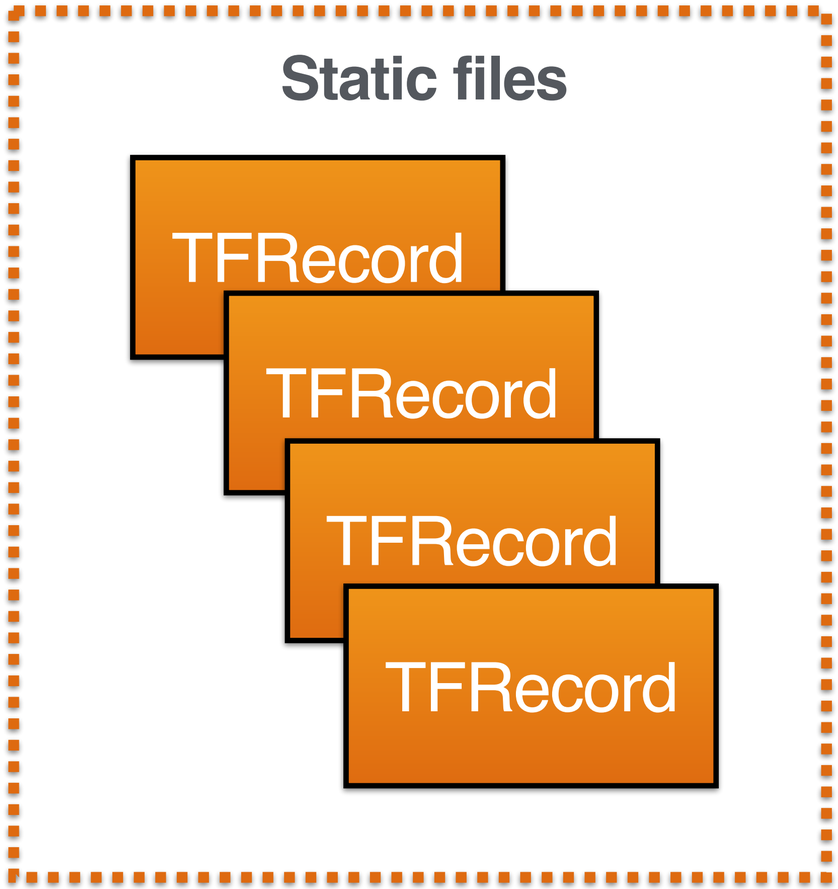


미리 문장 준비 및 구멍 뚫기를 해서 준비

동일한 Raw data에서 **n_duplicate** 만큼의

TrainingInstance를 생성해서 저장

기존 Static file 방식의 문제점



- 데이터 저장 공간이 많이 든다!
- Source Corpus (60G) x n_duplicate(10) x n_vocab(10)

  => **6TB**
- 모델이 중복된 TrainingInstance를 보게 된다.
- Preprocessing 방식이 바뀔 때마다 데이터를 새로 만들어야 한다!

  (vocab, tokenizer, max_seq_length, …)

보다 효율적인 데이터 파이프라인

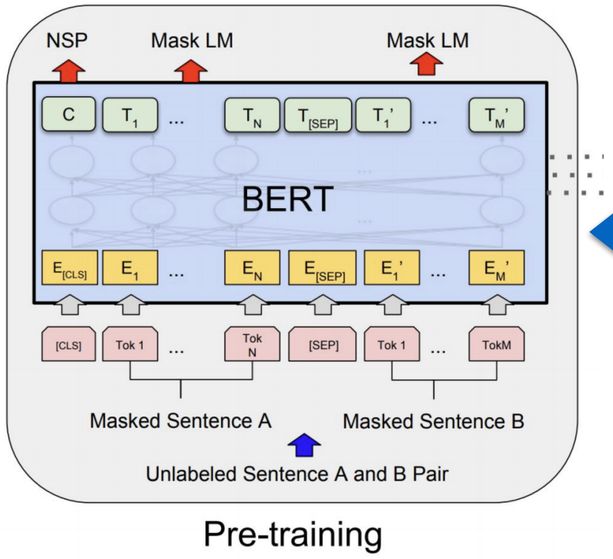
Static files

TFRecord

TFRecord

TFRecord

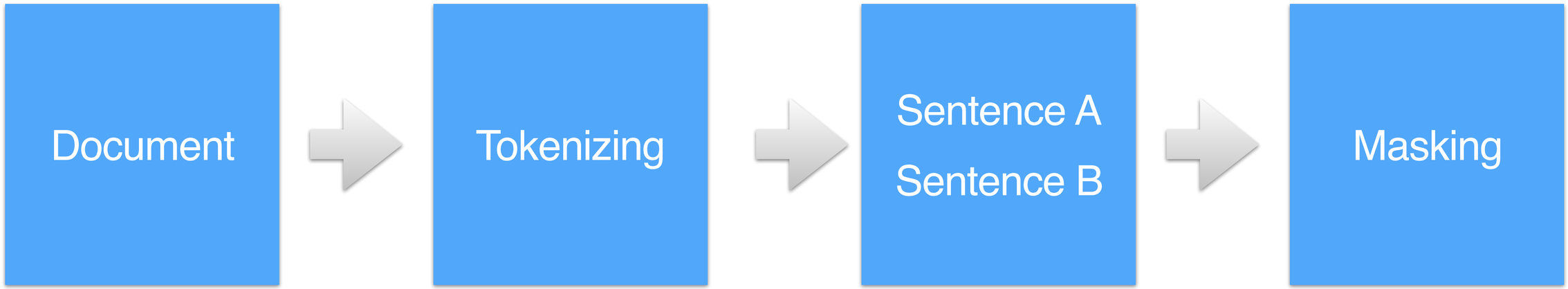
TFRecord



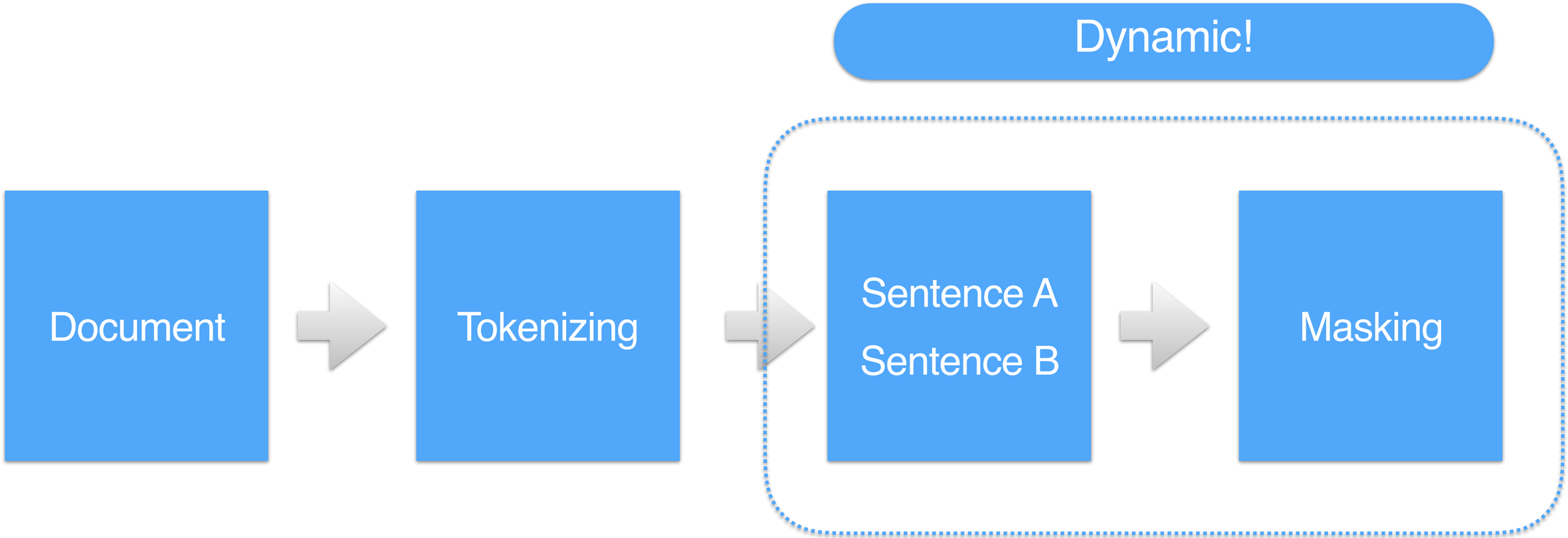
Pre-training

Dynamic Data Pipeline

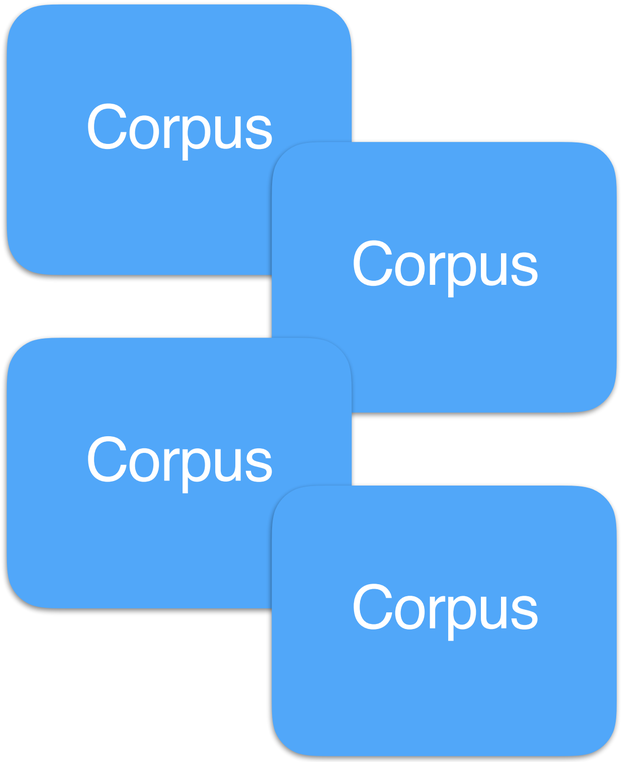
Dynamic Data Pipeline



Dynamic Data Pipeline



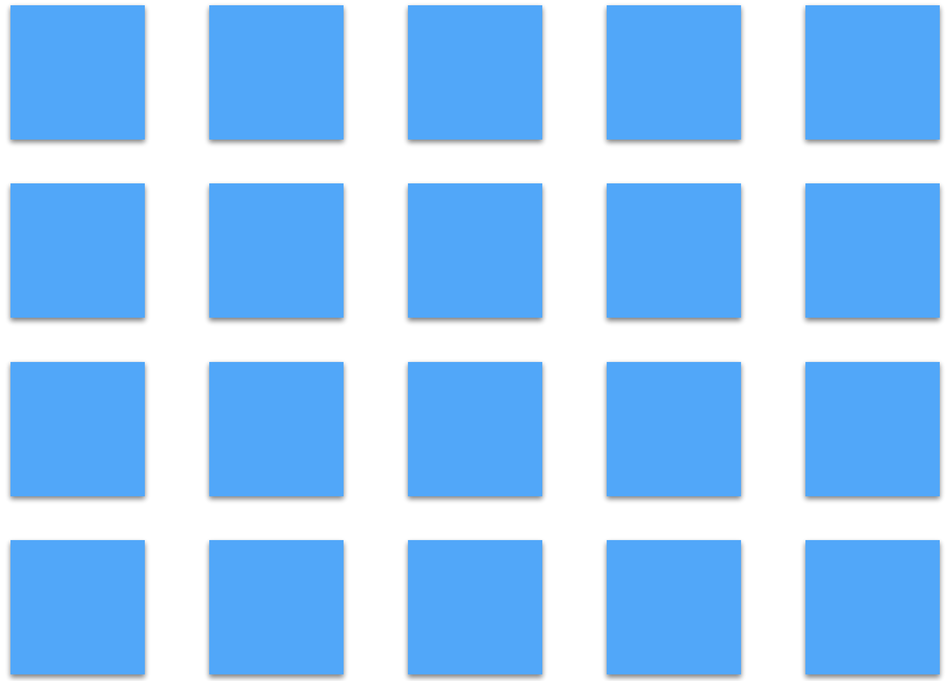
Dynamic Data Pipeline



Tokenizing 후,

비슷한 크기의 파일로

split! (200M ~ 300M)

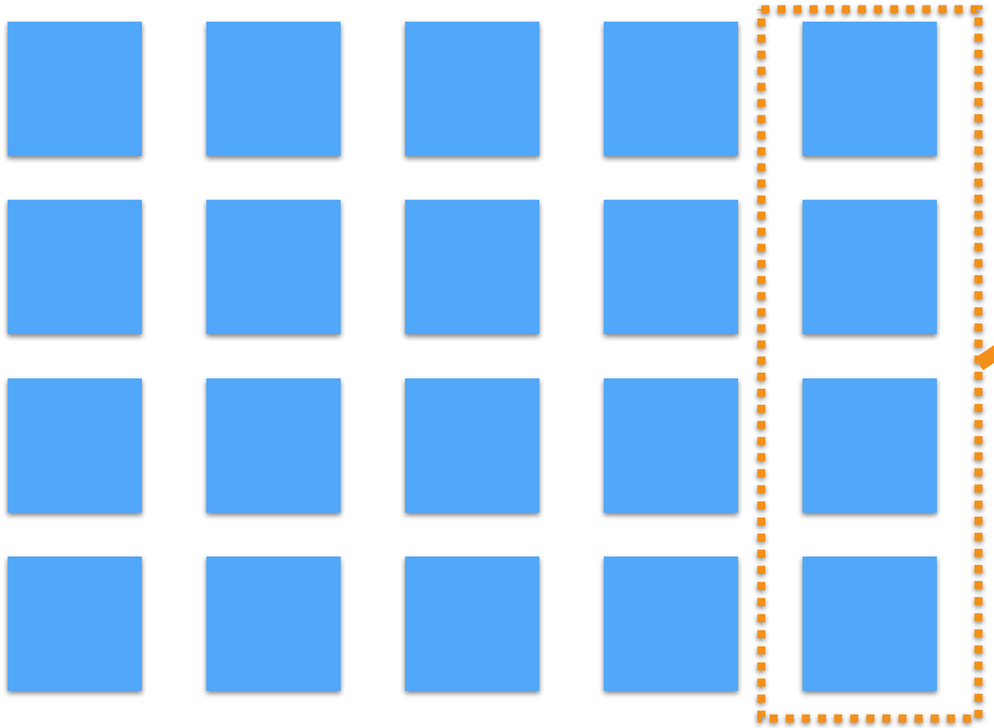


Dynamic Data Pipeline

파일 하나 != 트레이닝 인스턴스 하나

파일 하나에서는 N개(10k ~ 20k) 이상의 인스턴스가 만들어질 수 있음

Dynamic Data Pipeline



파일을 N개 씩 읽어서 동적으로

Training instance 준비

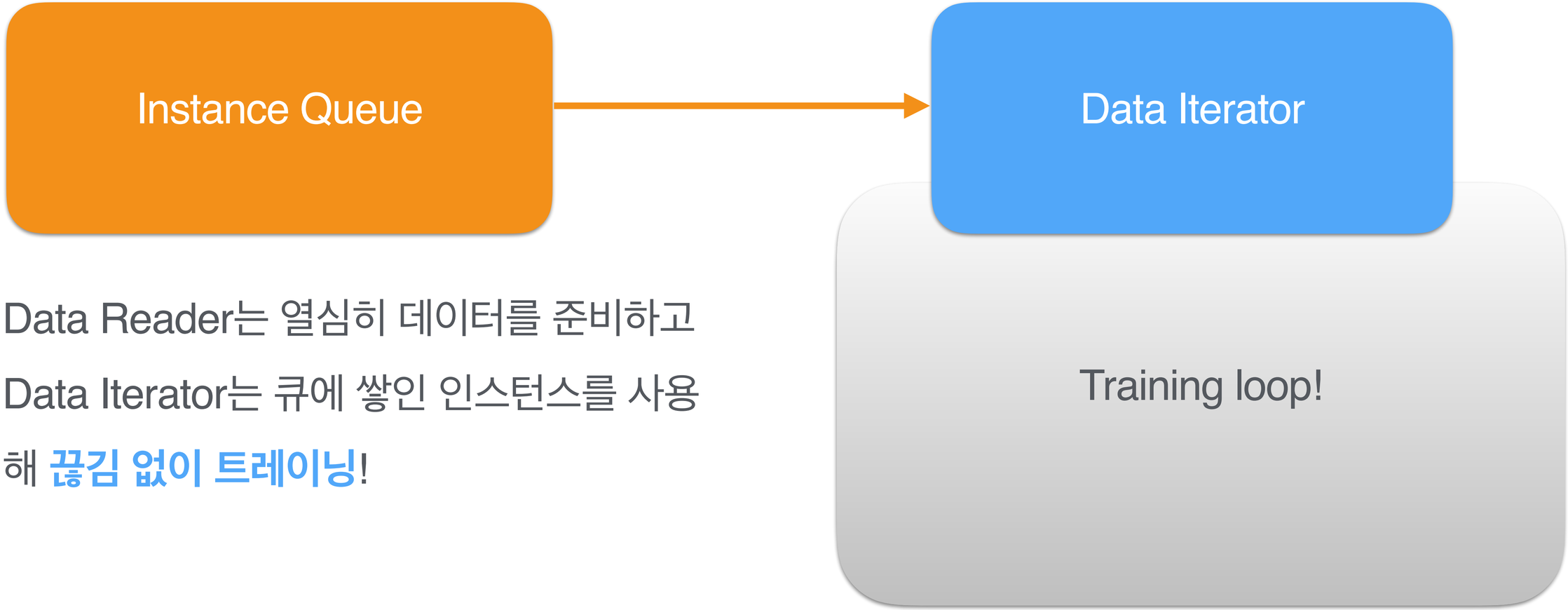
Data Reader

Sentence A & Sentence B

Masking

Instance Queue

Dynamic Data Pipeline



Data Reader는 열심히 데이터를 준비하고

Data Iterator는 큐에 쌓인 인스턴스를 사용

해 **끊김 없이 트레이닝**!

Dynamic Data Pipeline의 장점

- Source Corpus만 가지고 있으면 되기 때문에 저장 공간 절약 **(600GB -> 60GB)**
- 중복되는 TrainingInstance 없이 매번 다른 Mask를 만들어 낼 수 있음
  (아주 미미한 Generalization 효과)
- Training 중간에 Preprocessing option을 바꿀 수 있음 **(on-the-fly!!)**

저자의 Pre-training tip 중…

DEVIEW 2019

Attention의 연산 특성 상, Input sequence의 길이가 길어질수록 연산량이 Quadratic하게 많아 짐!

Longer sequences are disproportionately expensive because attention is quadratic to the sequence length. In other words, a batch of 64 sequences of length 512 is much more expensive than a batch of 256 sequences of length 128. The fully-connected/convolutional cost is the same, but the attention cost is far greater for the 512-length sequences. Therefore, one good recipe is to pre-train for, say, 90,000 steps with a sequence length of 128 and then for 10,000 additional steps with a sequence length of 512. The very long sequences are mostly needed to learn positional embeddings, which can be learned fairly quickly. Note that this does require generating the data twice with different values of `max_seq_length`.

총 100만 스텝 중, **90만 스텝은 1024B X 128L**로 진행하고, 나머지 **10만 스텝은 256B X 512L**로 해봐!

https://github.com/google-research/bert

저자의 Pre-training tip 중…

DEVIEW 2019

근데 max_seq_length가 128, 512인 데이터셋을 각각 만들어야 한다는 것 잊지말고!

Longer sequences are disproportionately expensive because attention is quadratic to the sequence length. In other words, a batch of 64 sequences of length 512 is much more expensive than a batch of 256 sequences of length 128. The fully-connected/convolutional cost is the same, but the attention cost is far greater for the 512-length sequences. Therefore, one good recipe is to pre-train for, say, 90,000 steps with a sequence length of 128 and then for 10,000 additional steps with a sequence length of 512. The very long sequences are mostly needed to learn positional embeddings, which can be learned fairly quickly. Note that this does require generating the data twice with different values of `max_seq_length`.

What ???!

60GB X 10 (n_duplicate) X 2 (128L, 512L) => **1.2T** (vocab 하나 당)

https://github.com/google-research/bert

On-the-fly preprocessing

트레이닝 중간에 max_seq_length, masked_lm_prob 등의 **옵션을 동적으로 변경**할 수 있음!

Variable Sequence Length : 90만 스텝까지는 1024배치 X 128 Length로

훈련 후, 남은 10만 스텝은 256배치 X 512 Length로 훈련 (훈련 속도 약 15% 증가)

Mask Scheduling : Masking의 난이도를 점진적으로 올릴 수 있음

(Basic Masking => Ngram masking ratio를 늘려나가기)

Variable Sequence Length

1024B x 128L

VS

256B x 512L

한정된 자원 하(**V100 X 8**)에서

- 성능 저하 없이, 전체 트레이닝 속도를 **약 15%** 올릴 수 있음
- **짧은 문장이 주인 Text Classification** 다운스트림 테스크에서 더 큰 배치 사용으로 인한 **성능 향상**

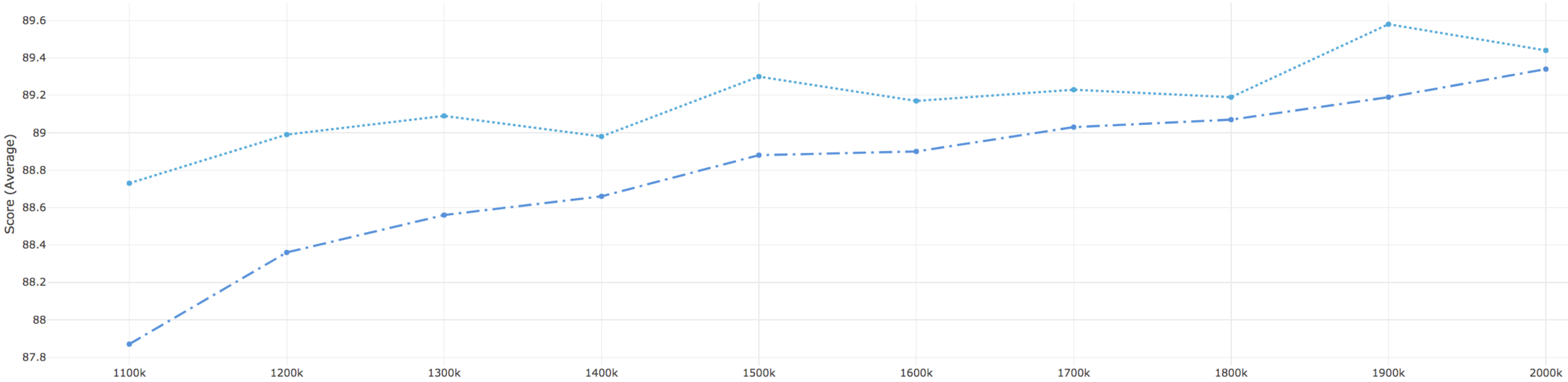
Mask Scheduling

DEVIEW 2019

K step마다 전체 Mask 중
Ngram masking의 비율을 점진적으로 늘려 나감

Mask Scheduling
Basic Mask 1M 이후 Ngram Masking 1M

KLUE Benchmark 8개의 평균 점수



NSP가 필요없다고?!

최근 여러 논문에서 NSP의 필요성에 대한 의문을 던짐

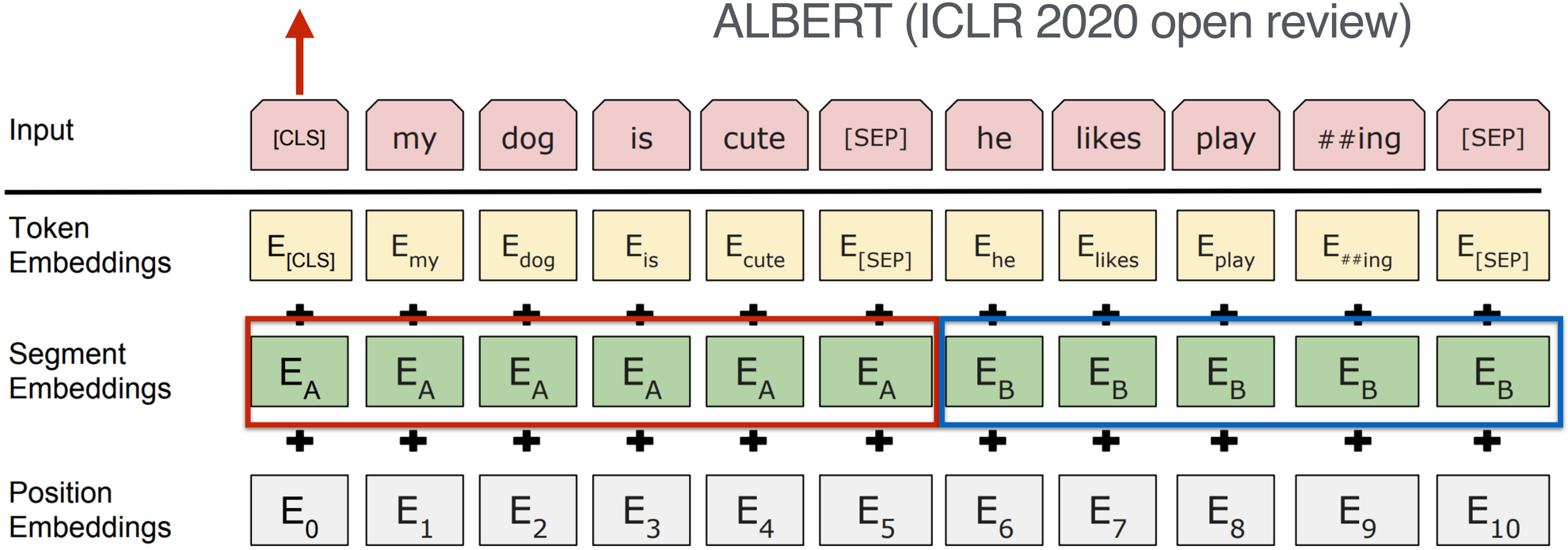
(MLM에 비해 너무 쉬워서... 차라리 긴 시퀀스를 보는 것이 더 도움!)

SpanBERT (Joshi et al, 2019)

RoBERTa (Liu et al, 2019)

ALBERT (ICLR 2020 open review)

Next Sentence Prediction ?!



이미지: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding - 2018 J Devlin

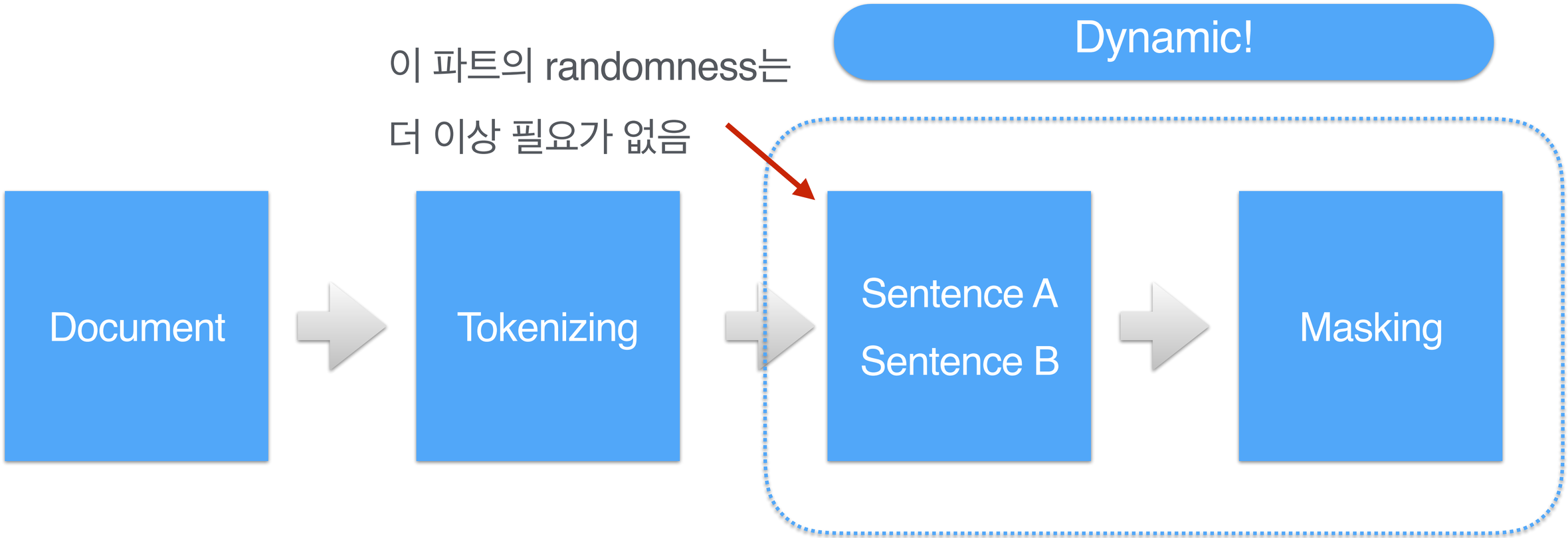
FULL SENTENCE로 준비하자!

SENTENCE PAIR + NSP (기존 방식)

Random text 없이 **문서 내의 연속된 시퀀스**로 인스턴스를 채운다!
다만, 문서가 바뀔 때만 Seperator를 붙이고 다시 이어서 채움

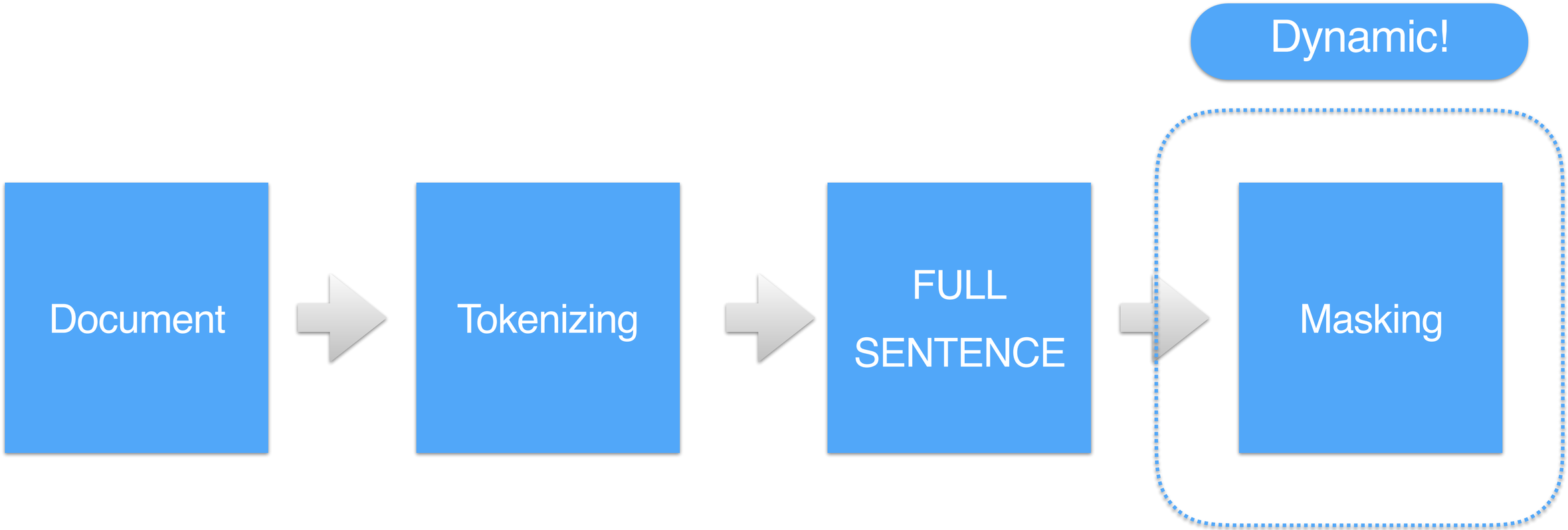
FULL SENTENCE

기존의 Dynamic Data Pipeline

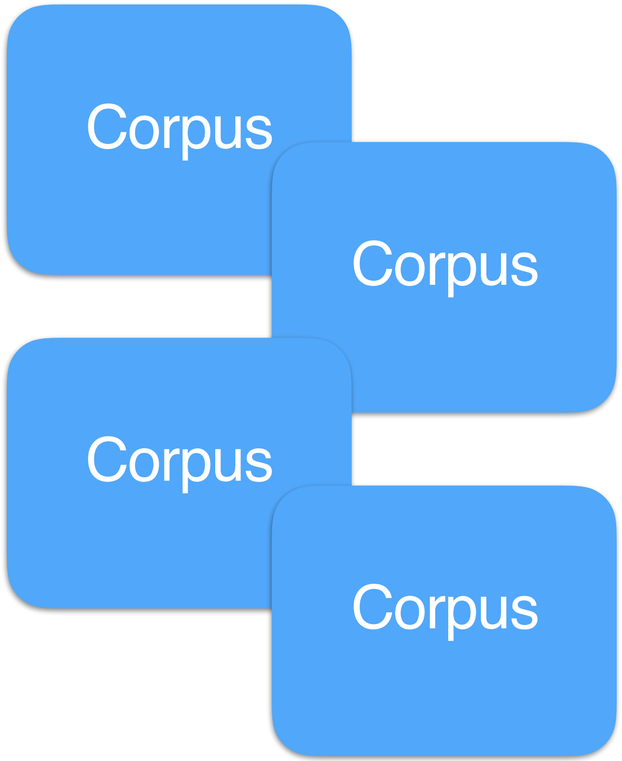


새로운 Dynamic Data Pipeline

DEVIEW 2019



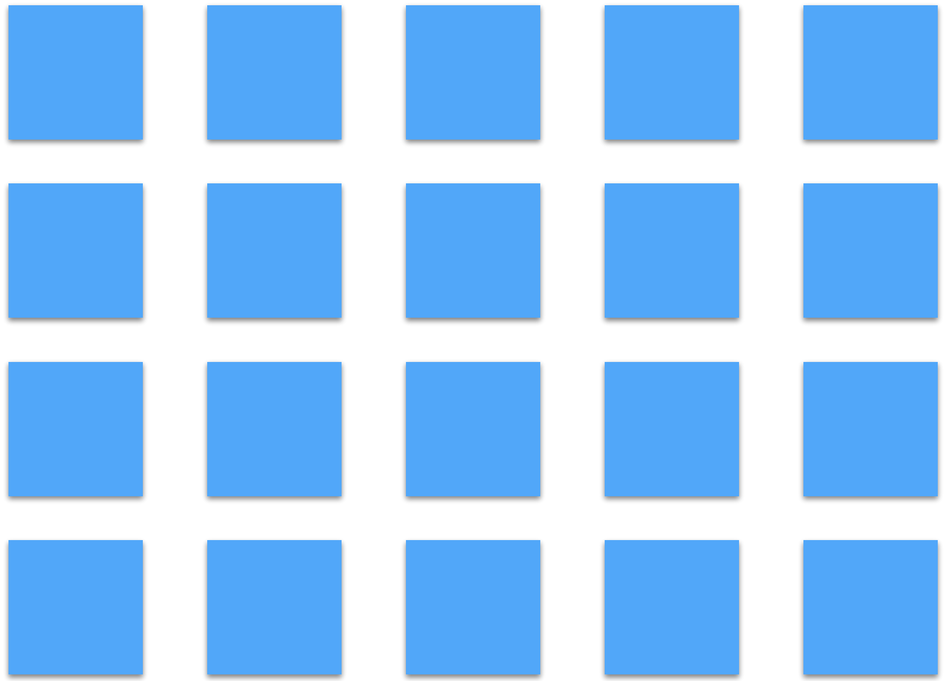
새로운 Dynamic Data Pipeline



Tokenizing 후,

FULL SENTENCE

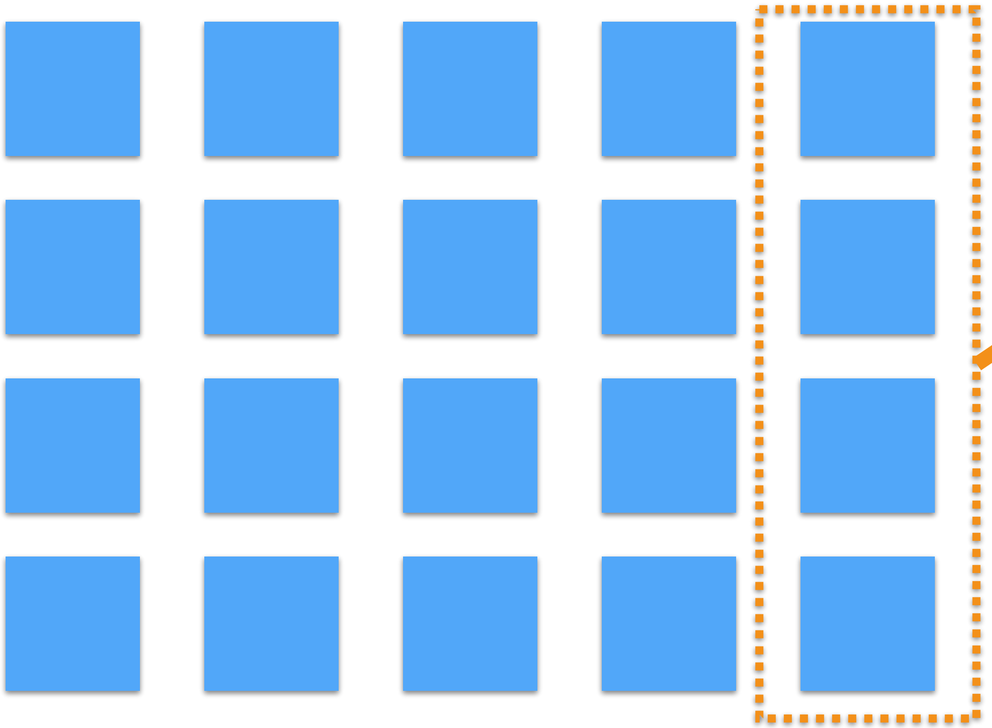
형태로 준비(**indexing**)



새로운 Dynamic Data Pipeline

파일 하나 == 트레이닝 인스턴스 하나

새로운 Dynamic Data Pipeline



파일을 Batch size만큼

읽어서 Training instance 준비

Data Iterator

Masking

+ NSP or SOP

Training loop

두 파이프라인의 비교

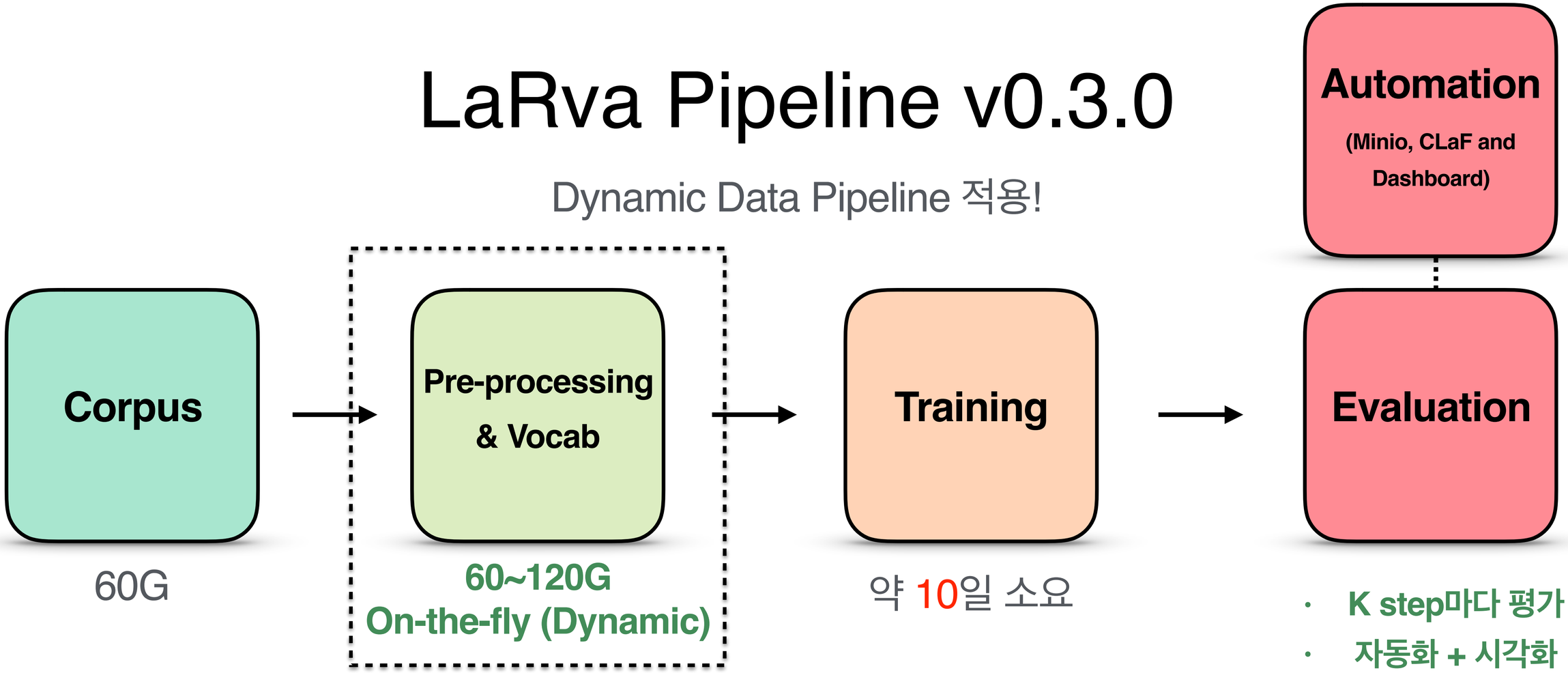
Roberta BASE (256B 기준)	Hierarchical Data Loader (2 step pipeline)	Flat Data Loader (FULL SENTENCE)
# Sample/s (avg)	280	350
Variable Sequence Length	가능	불가능
Mask Scheduling	가능	가능
Indexed 파일 필요 여부 (Storage x 2)	불필요	필요
Distributed training	불가능(까다로움..)	가능

(BERT Base 기준)

DEVIEW 2019

LaRva Pipeline v0.3.0

Dynamic Data Pipeline 적용!

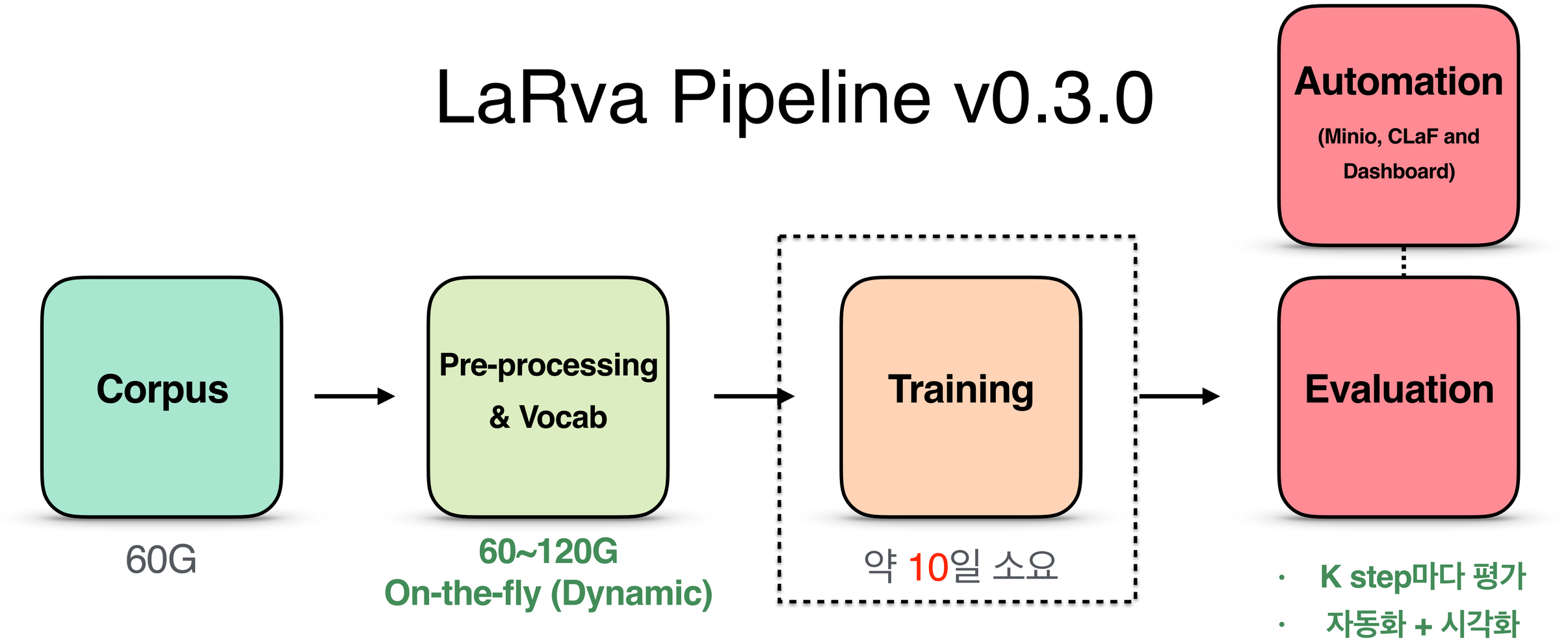


2-3. 학습을 빠르게

(BERT Base 기준)

DEVIEW 2019

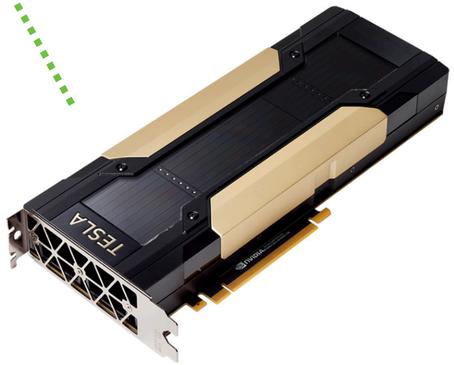
LaRva Pipeline v0.3.0



메모리와의 사투..

BERT Base 256 batch를 학습시키려면..

GPU 하나 당 32 Batch씩 X 8 (GPU) = 256



NVIDIA V100 32G

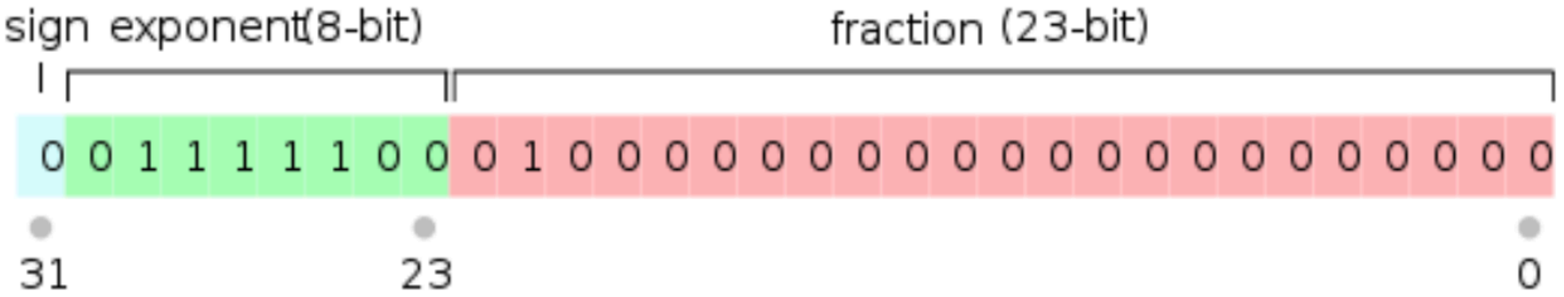
하지만 Single Precision (FP32) 기준으로는

GPU 하나 당 24 Batch가 최대…!

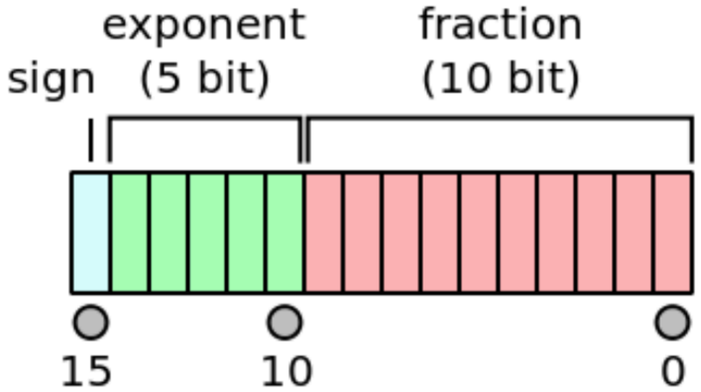
(24 Batch x 8 GPU = 192)

Single, Half Precision

Single Precision



Half Precision

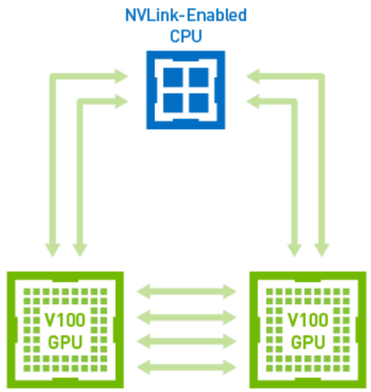


- Memory 절감
- 연산 속도 증가 (**약 2배**)

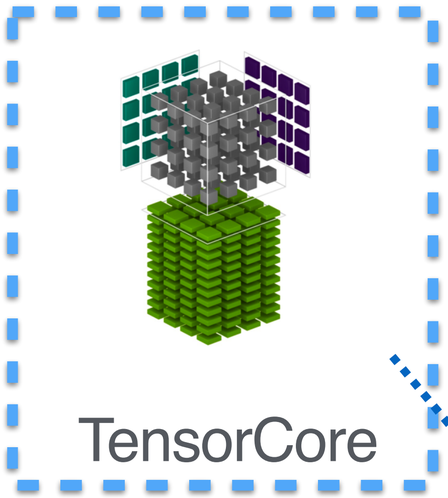
== **더 큰 배치로 더 빨리** !

source: https://en.wikipedia.org/wiki/Half-precision_floating-point_format

Let’s Mixed Precision Training !!



NVLink



TensorCore

+

FP16_Optimizer

With Dynamic LossScaler

NVIDIA V100 32G

FP16 및 FP32 부동소수점 연산에

최적화된 아키텍처

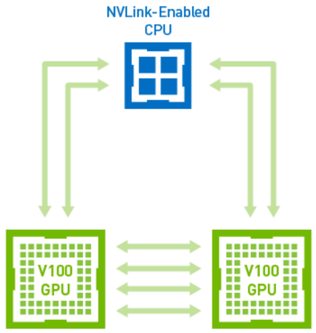
source: https://www.nvidia.com/ko-kr/data-center/tensorcore/
source: https://www.nvidia.com/ko-kr/data-center/nvlink/

Let’s Mixed Precision Training !!

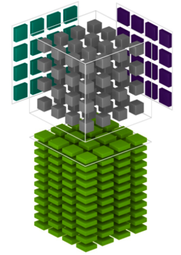
DEVIEW 2019

BERT Base 256 batch 기준!

- 32 Batch x 8 GPU = 256 Batch 실현(192B => **256B**)
- 성능 하락 없이 약 2배 정도의 속도 향상 (**10일 => 6일**)
- 모델 저장 용량도 **절반**



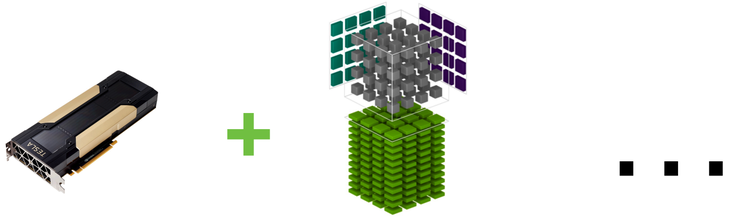
NVLink



TensorCore

NVIDIA V100

더 큰 사이즈의 모델을 학습 시키려면?



BERT Large는 Mixed Precision을 사용해도

GPU(V100) 당 16 Batch가 올라감

16 Batch x 8 GPU => **128 batch**…

약 **3 ~ 4주** 소요...

Gradient Accumulation x 2

128 x 2 => 256

심지어 최신 모델들의 배치사이즈는..

XLNet

Hparam	Value
Number of layers	24
Hidden size	1024
Number of attention heads	16
Attention head size	64
FFN inner hidden size	4096
Dropout	0.1
Attention dropout	0.1
Partial prediction K	6
Max sequence length	512
Memory length	384
Batch size	2048
Learning rate	1e-5
Number of steps	500K
Warmup steps	20,000
Learning rate decay	linear
Adam epsilon	1e-6
Weight decay	0.01

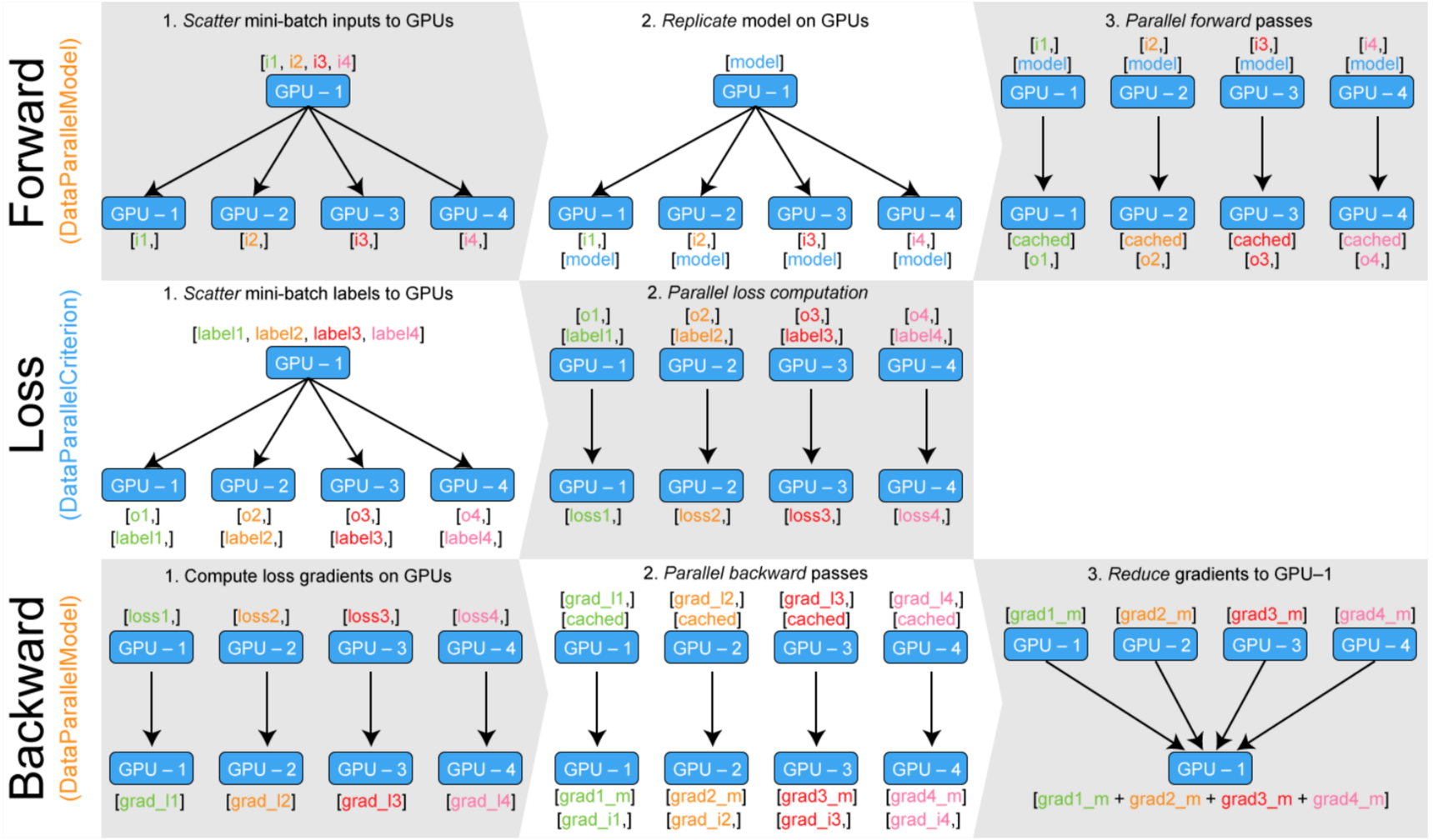
RoBERTa

Hyperparam	$\text{RoBERTa}_{\text{LARGE}}$	$\text{RoBERTa}_{\text{BASE}}$
Number of Layers	24	12
Hidden size	1024	768
FFN inner hidden size	4096	3072
Attention heads	16	12
Attention head size	64	64
Dropout	0.1	0.1
Attention Dropout	0.1	0.1
Warmup Steps	30k	24k
Peak Learning Rate	4e-4	6e-4
Batch Size	8k	8k
Weight Decay	0.01	0.01
Max Steps	500k	500k
Learning Rate Decay	Linear	Linear
Adam ϵ	1e-6	1e-6
Adam β_1	0.9	0.9
Adam β_2	0.98	0.98
Gradient Clipping	0.0	0.0

source: XLNet: Generalized Autoregressive Pretraining for Language Understanding (2019, Yang et al.)

RoBERTa: A Robustly Optimized BERT Pretraining Approach (2019, Liu et al.)

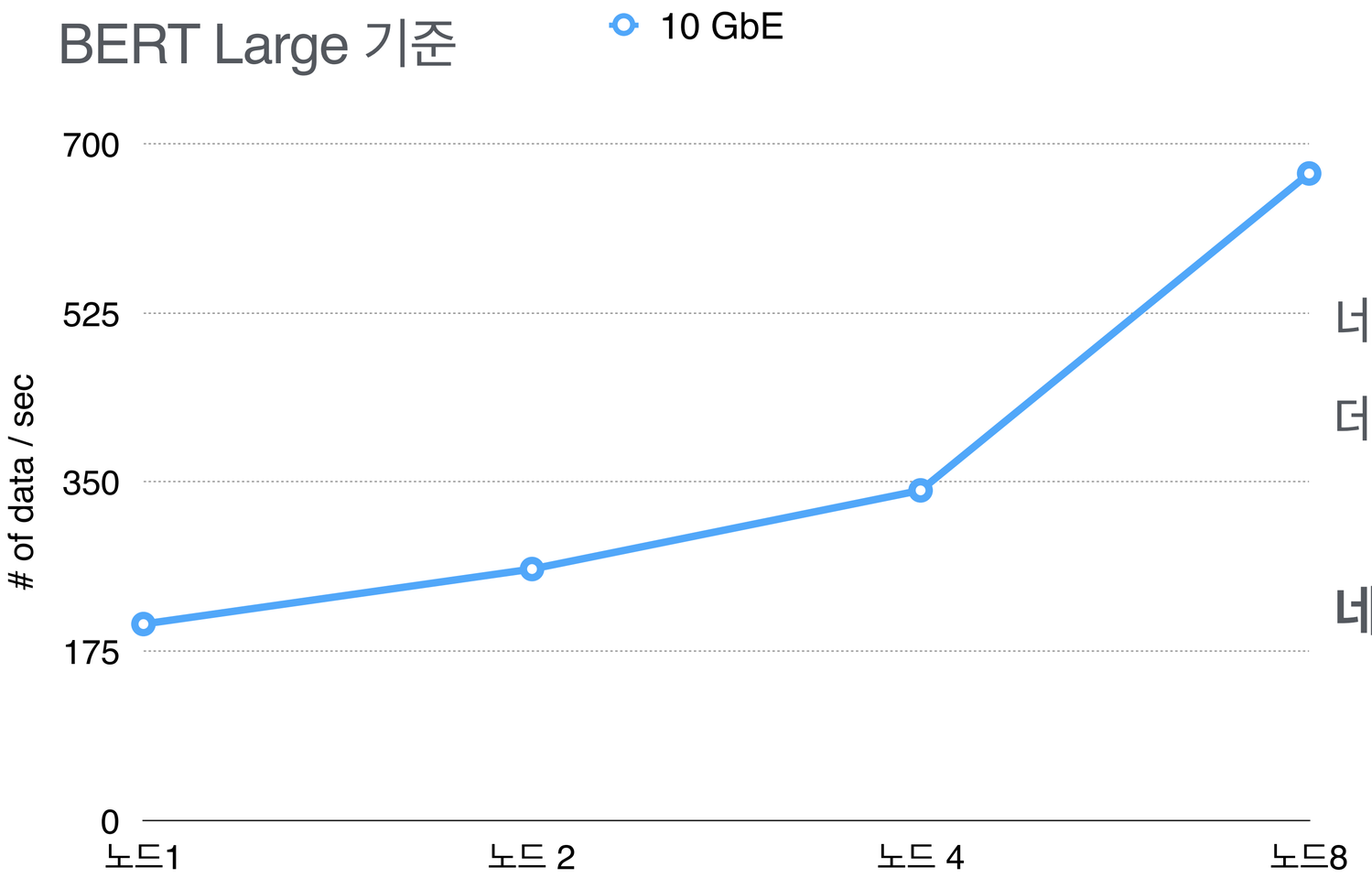
Let's DistributedDataParallel !



source: https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255

단순하게 노드만 늘렸더니..

DEVIEW 2019



네트워크 I/O Bottleneck으로 인해

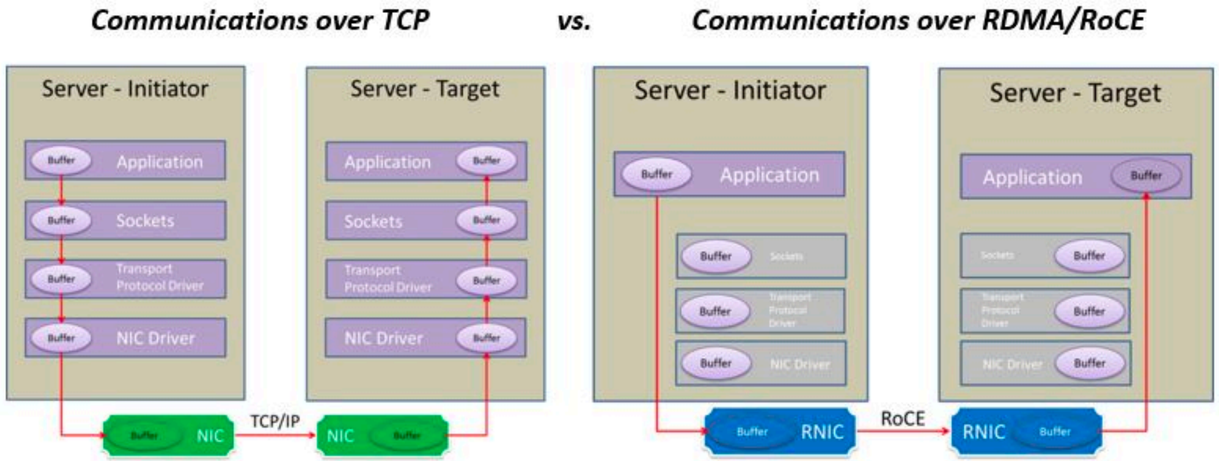
데이터 처리량의 증가폭이 너무 비효율적...

네트워크도 엔지니어링을 해보자..!

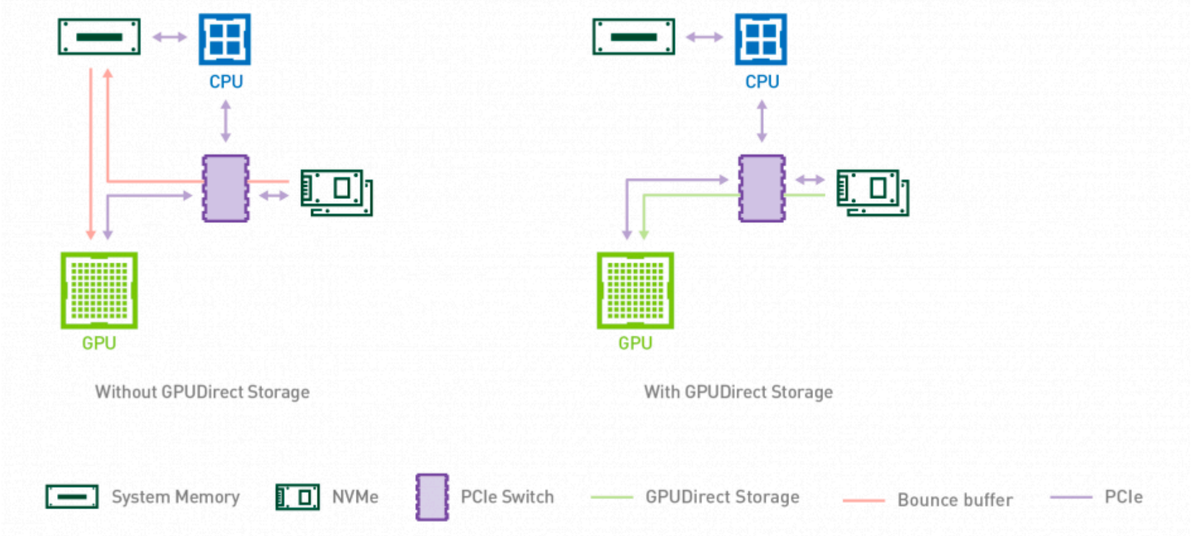
RDMA + GPU Direct

단일 노드를 너머 멀티노드 학습을 진행할때, NCCL의 네트워크 튜닝도 중요!

RDMA: TCP/IP와는 달리 불필요한 protocol overhead를 줄이고, infiniband NIC가 CPU를 거치지 않고 직접 메모리에 통신을 수행하여 불필요한 복사나 과도한 CPU개입 회피
GPU Direct RDMA: 원격지 노드의 GPU 간의 통신시에 GPU가 CPU와 host 메모리를 거치지 않고 데이터를 전송 (nv_peer_mem 설치 필요)



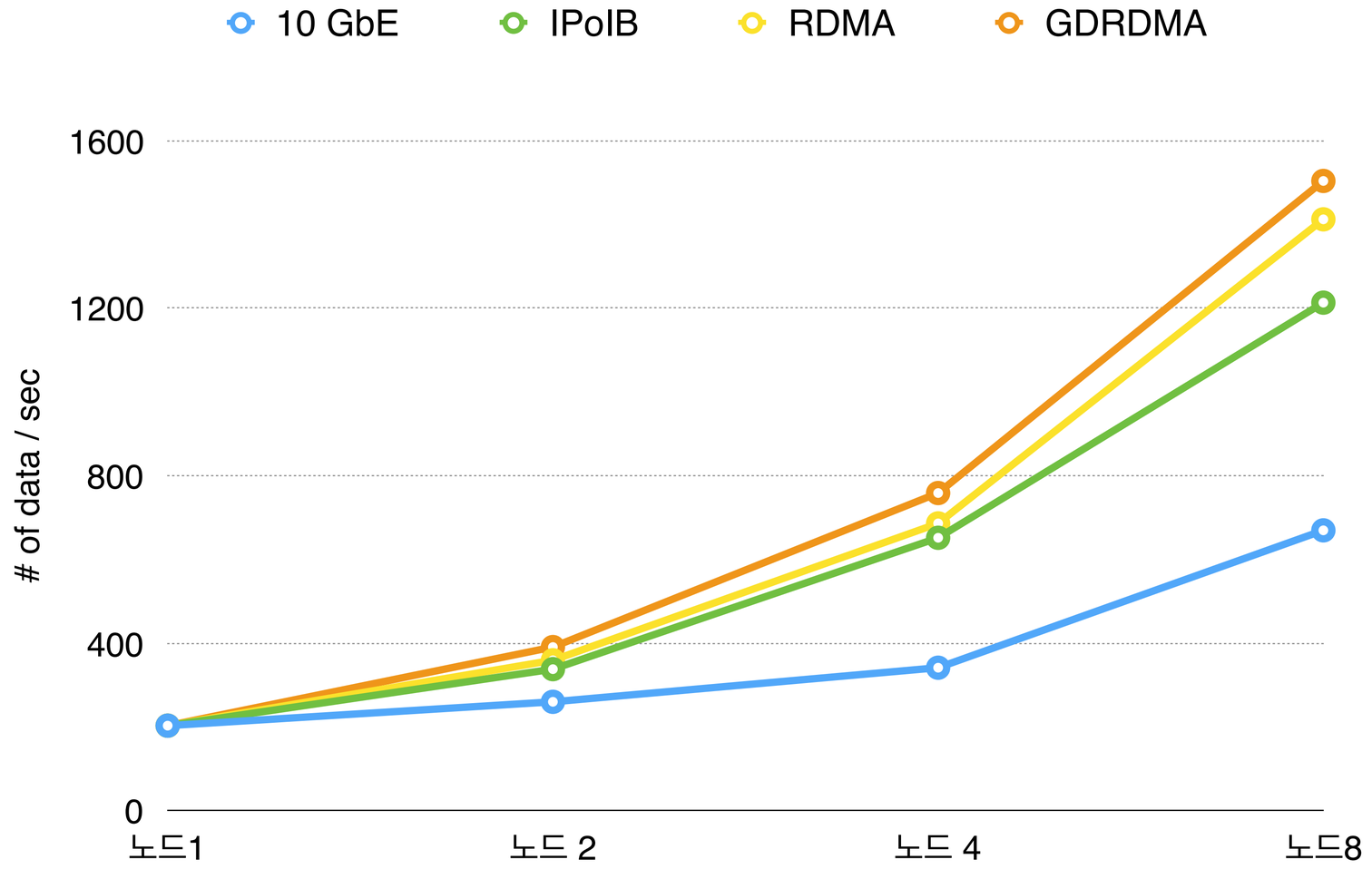
source: Mellanox



source: https://developer.nvidia.com/gpudirect

네트워크 속도 벤치마크

BERT Large 기준



DDP 적용 결과

BERT **Large** model Pre-training (**256B**)

DataParallel & Gradient Accumulation

(NVIDIA V100 1 Node)

=> 약 **25일** 소요

VS

DistributedDataParallel

(NVIDIA V100 2 Node)

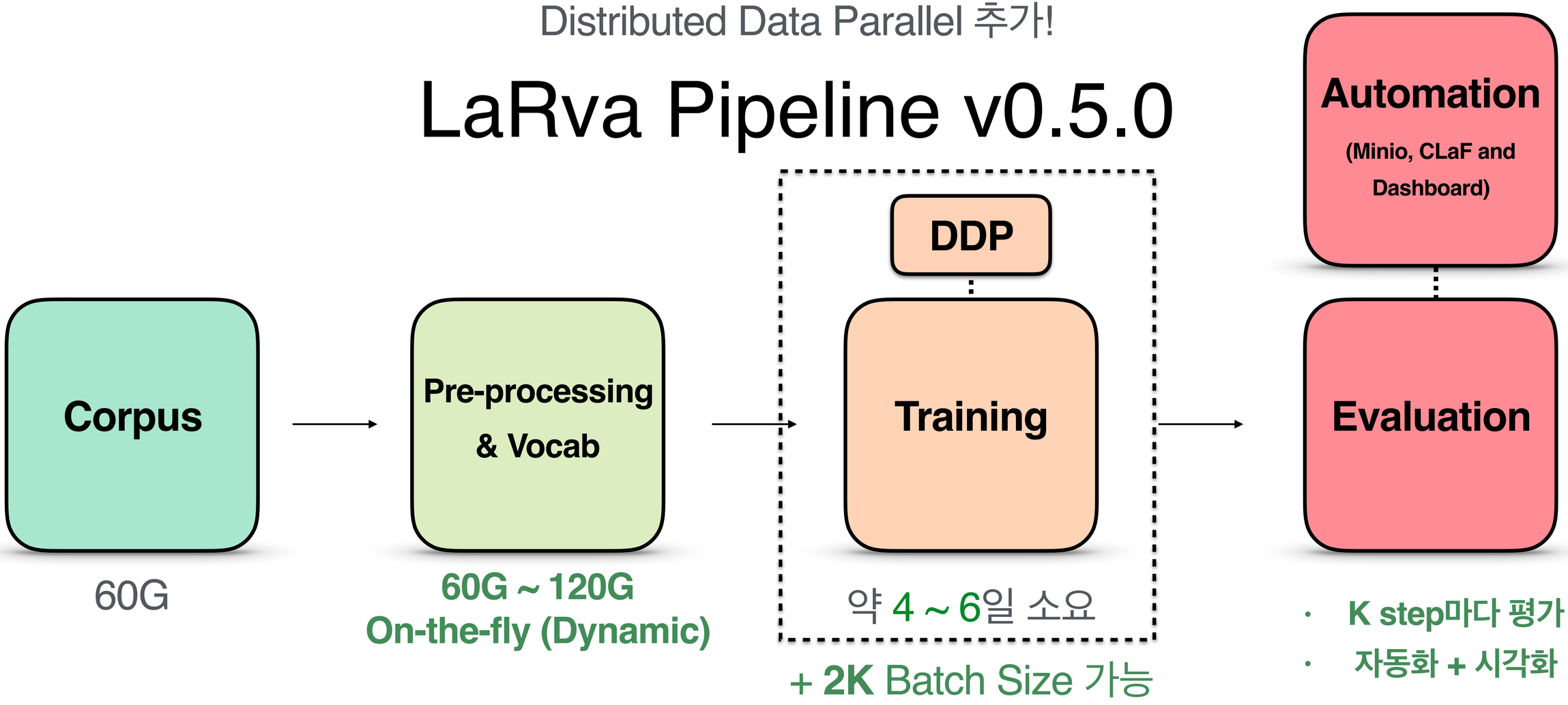
=> 약 **10일** 소요

더 적은 Step만으로도 (더 좋은) 성능 도달!

BERT LARGE	256B	1024B
스텝수	1M	500K
사용 자원	V100 x 16	V100 x 64
소요 기간	약 10일	**약 6일**

Distributed Data Parallel 추가!

LaRva Pipeline v0.5.0



3. 더 좋은 언어 모델을 위해

KLUE: 한국어 이해 평가 벤치마크

KLUE Benchmark

	HQA	HQA-long	KorQuAD	MR-2	MR-5	QS	SM	PAWS
Task 종류	기계독해 (역사)	기계독해 (역사)	기계독해 (Wiki)	감정분석 (영화리뷰)	감정분석 (영화리뷰)	유사한 문장	유사한 문장	유사한 문장 (단어 섞음)
Metric	EM/F1	EM/F1	EM/F1	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy
공개 여부			Y	Y				Y

KorQuAD 1.0

Naver sentiment movie corpus v1.0

Google AI

KorQuAD 1.0: https://korquad.github.io/category/1.0_KOR.html,
nsmc: https://github.com/e9t/nsmc/
PAWS: https://github.com/google-research-datasets/paws

Masking Strategy

Sentence	Harry	Potter	is	a	series	of	fantasy	novels	written	by	British	author	J.	K.	Rowling
Basic-level Masking	[mask]	Potter	is	a	series	[mask]	fantasy	novels	[mask]	by	British	author	J.	[mask]	Rowling
Entity-level Masking	Harry	Potter	is	a	series	[mask]	fantasy	novels	[mask]	by	British	author	[mask]	[mask]	[mask]
Phrase-level Masking	Harry	Potter	is	[mask]	[mask]	[mask]	fantasy	novels	[mask]	by	British	author	[mask]	[mask]	[mask]

데이터 전처리에 추가적인 파이프라인이 필요함

(NER 모델, 구문 분석기 등)

=> Space-level masking! (어절 단위)

출처: ERNIE: Enhanced Representation through Knowledge Integration - 2019 Y Sun

Masking Strategy Example

Basic-level Masking

```
tokens: [CLS] 지 ##쿠 ##호 ##노가 [MASK] 역 [SEP] 지 ##쿠 ##호 ##노가 ##타 역 ( ) 은 일본 [MASK] ##르트 ##현 노 ##가 ##타 시에 위치
한 지 ##쿠 ##호 전기 철도 [MASK] ##쿠 ##호 전기 [MASK] ##선의 [MASK] ##역이 ##다 . 이 노선 ##의 종 ##착 ##역이 [MASK] , 역 번호 ##는
C ##K 21 ##이며 , [MASK] ##식 승강 [MASK] 2 ##면 2 ##선의 구조를 [MASK] 지상 ##역이 ##다 [MASK] [SEP]
segment_ids: 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
is_random_next: False
masked_lm_positions: 5 18 19 31 35 37 47 57 60 66 70
masked_lm_labels: ##타 후쿠 ##오카 지 철도 철도 ##자 상대 ##장 갖춘 .
```

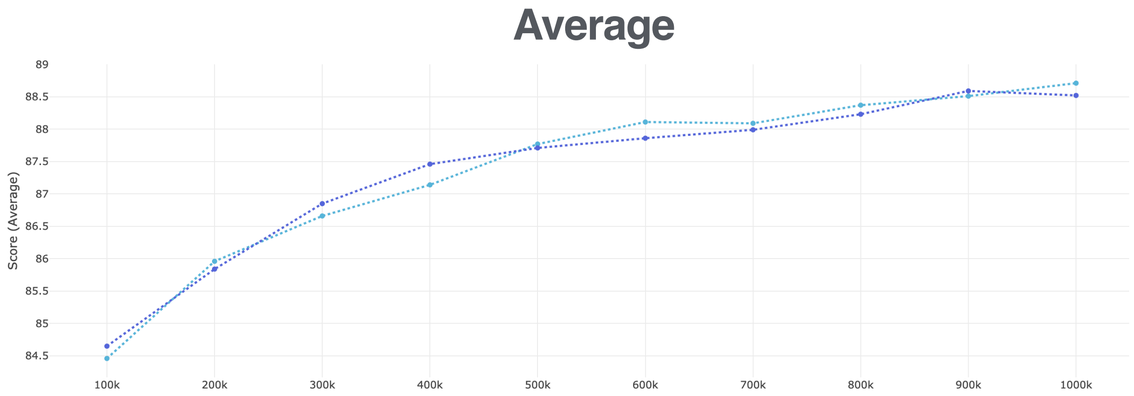
Space-level Masking

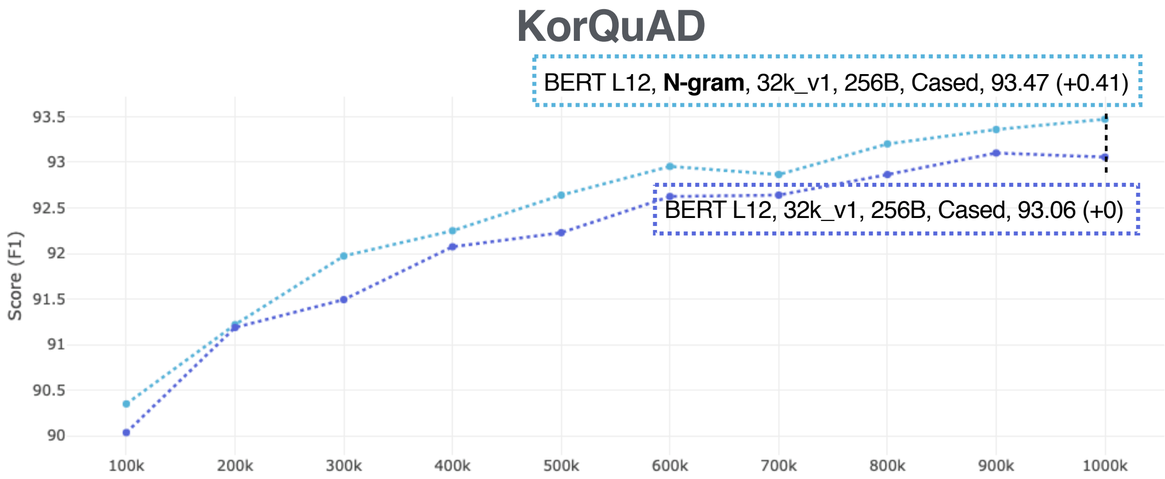
```
tokens: [CLS] 지 ##쿠 ##호 ##노가 ##타 역 [SEP] 지 ##쿠 ##호 ##노가 ##타 역 ( ) 은 일본 후쿠 ##오카 ##현 노 ##가 ##타 시에 위치한 지 #
#쿠 ##호 전기 철도 지 ##쿠 ##호 전기 철도 ##선의 [MASK] [MASK] ##다 . 이 노선 ##의 [MASK] [MASK] [MASK] 프란 , 역 번호 ##는 C ##K
[MASK] [MASK] , 상대 ##식 승강 ##장 2 ##면 2 ##선의 구조를 갖춘 [MASK] [MASK] [MASK] . [SEP]
segment_ids: 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
is_random_next: False
masked_lm_positions: 37 38 44 45 46 47 54 55 67 68 69
masked_lm_labels: 철도 ##역이 종 ##착 ##역이 ##자 21 ##이며 지상 ##역이 ##다
```

Space-level Masking Strategy

Space-level Masking은 모든 Task기준 (Average)으로는 미미한 증가가 있다.

그러나, KorQuAD 와 같은 기계독해 Task에는 효과가 크다.



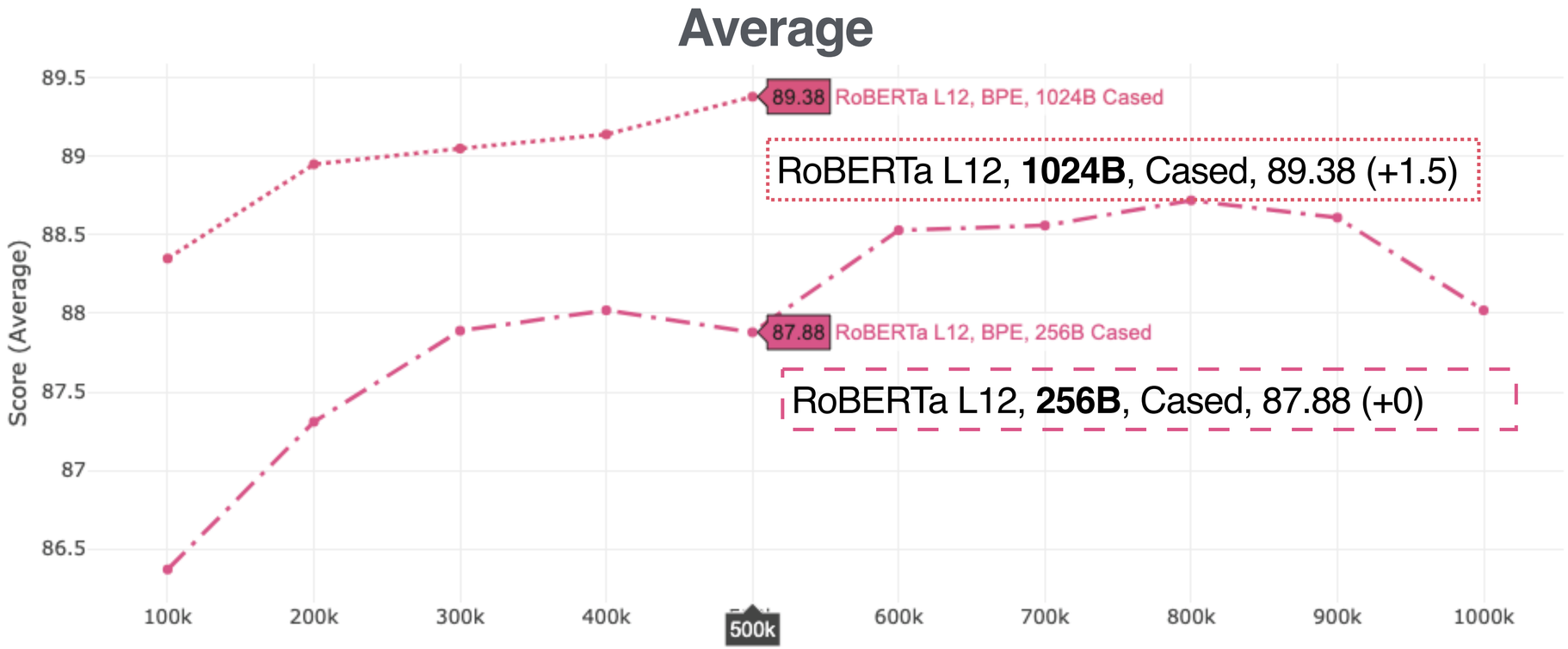


Google Whole Word Masking과 동일

Batch Size: 256 vs 1024

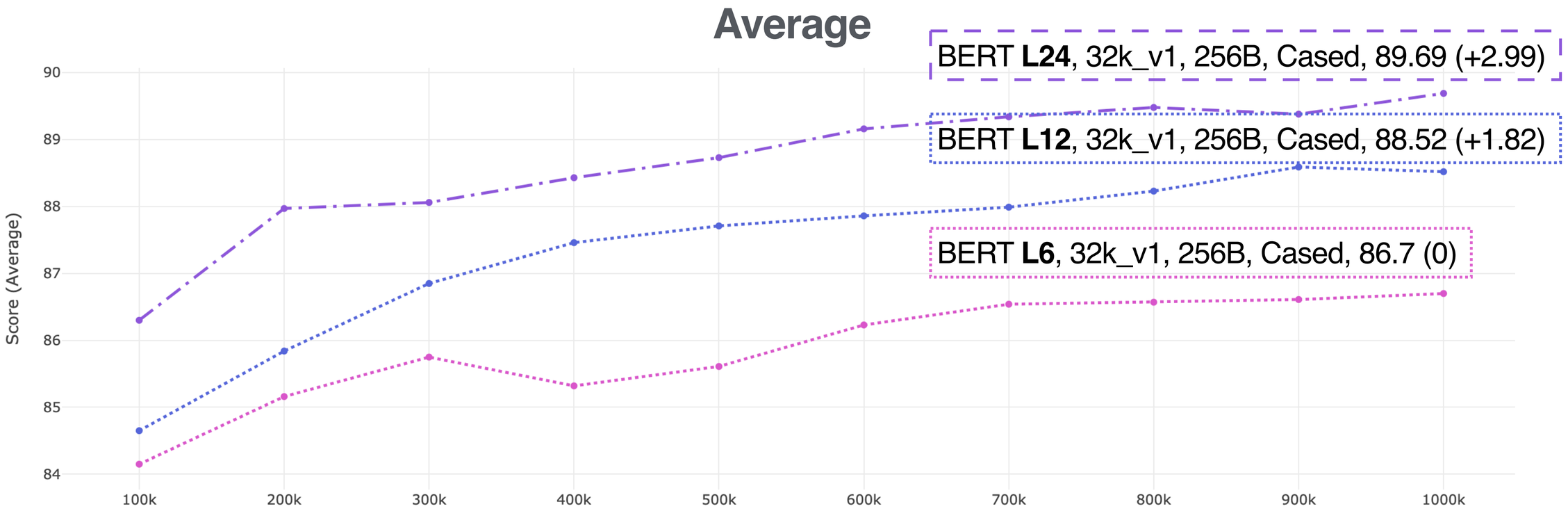
배치 사이즈는 모든 Task에 대해서 클 수록 성능이 올라간다.

(최신 논문들의 BatchSize는 4K, 8K)



Model Size: Bigger is Better

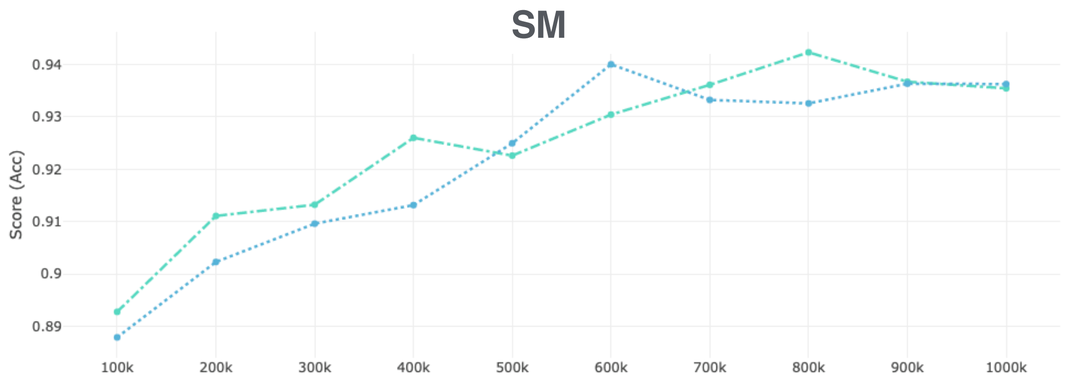
모델의 사이즈는 모든 Task에 대해서 클 수록 성능이 올라간다.

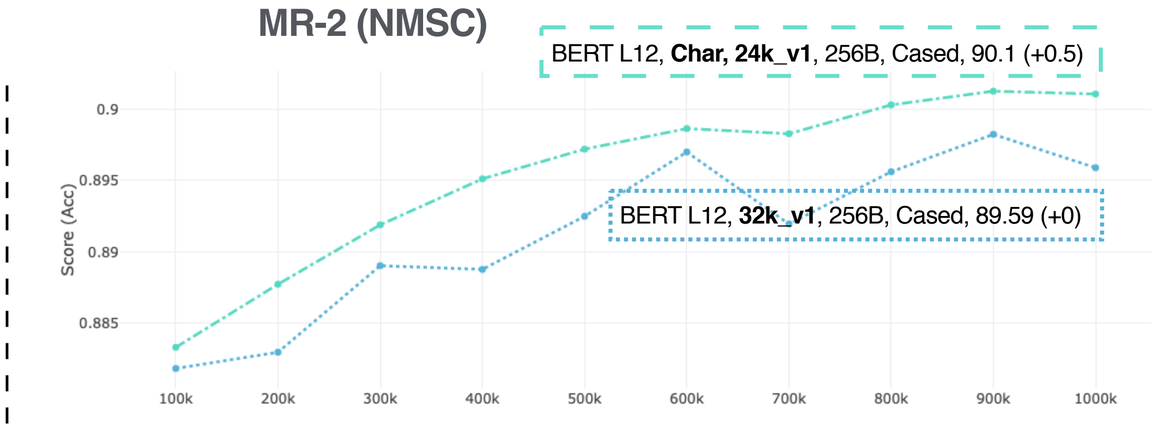


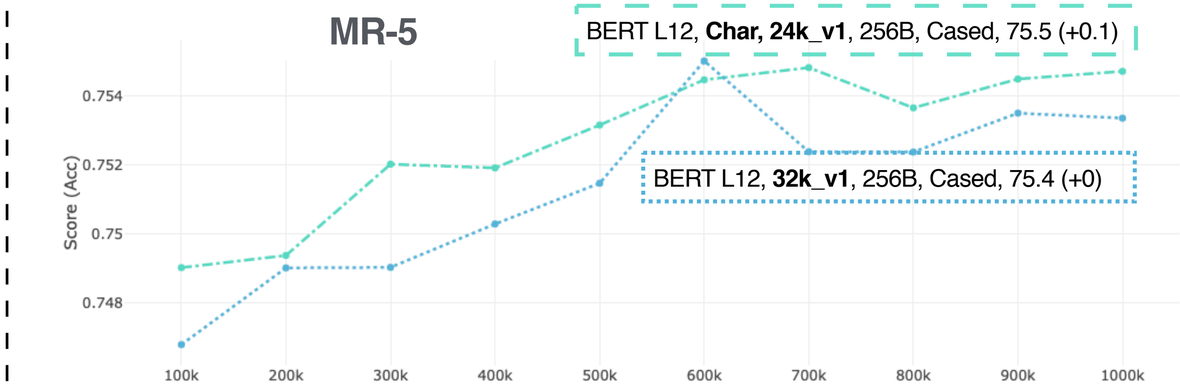
Tokenizer: Char vs Sub-word

감정분석에서는 Char 단위의 성능이 좋으나,

그외 Task에서는 비슷한 성능





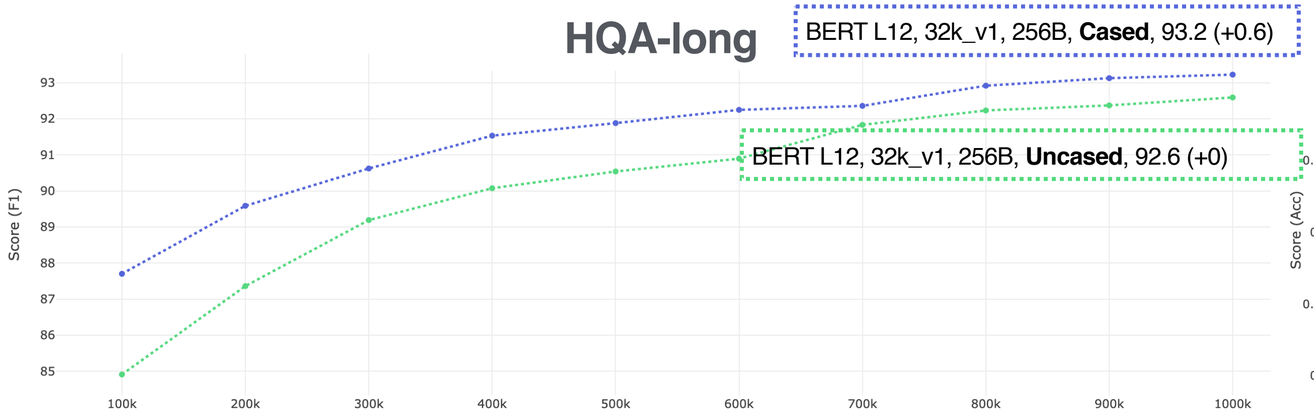


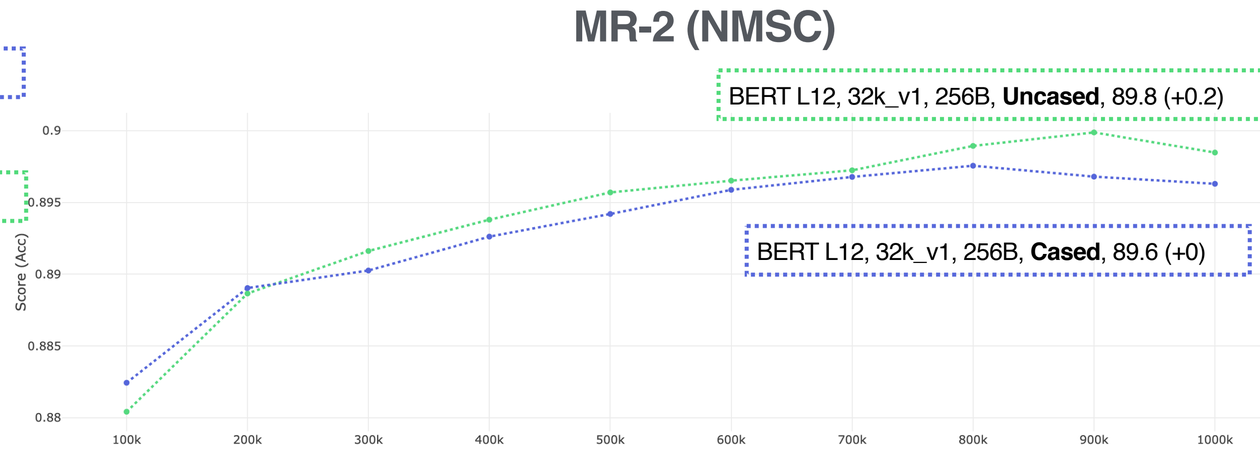
Tokenizer: Cased vs Uncased

DEVIEW 2019

대체로 성능은 비슷하나, 자모 분해로 Token 수가 늘어나 기계독해 성능 하락 Cased 우세

감정분석 Task에서는 Uncased이 약간 우세





* Cased 와 Uncased 차이 예시

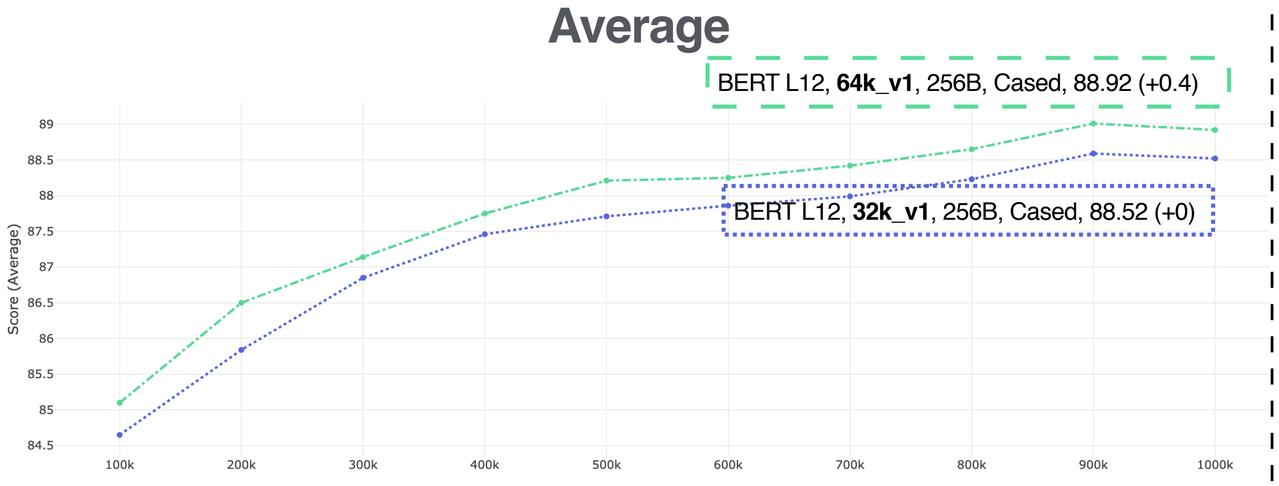
Input: 안녕하세요

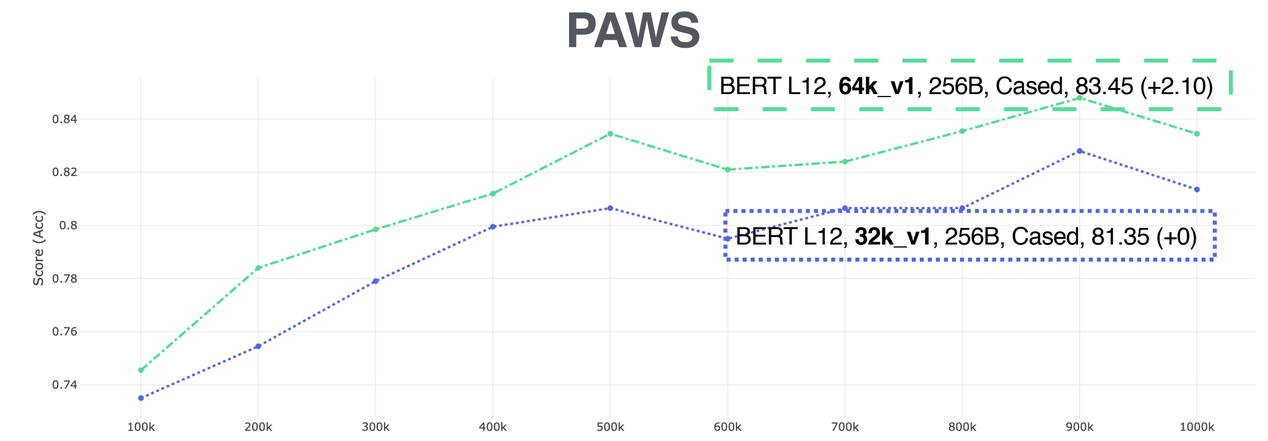
- (bert-base-multilingual-uncased, do_lower_case=True), ['ㅇ', '## ㅏㄴ', '##녀', '##ㅇ', '##하', '##세', '##요']
- (bert-base-multilingual-cased, do_lower_case=False), ['안', '##녕', '##하', '##세', '##요']

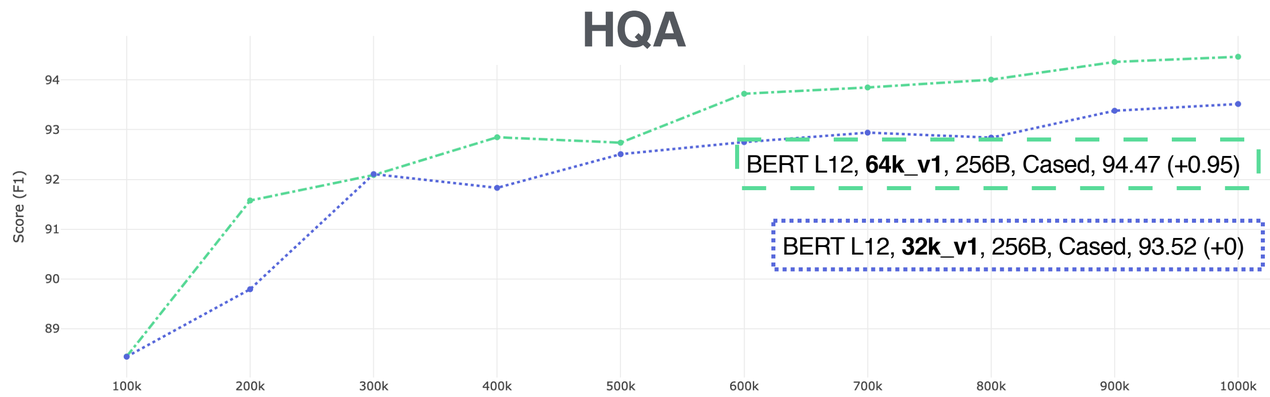
Vocab Size: 32k vs 64k

DEVIEW 2019

64k가 기계독해, PAWS Task에서 큰 성능 향상을 보여주면서, Average 또한 상승 (+0.4)



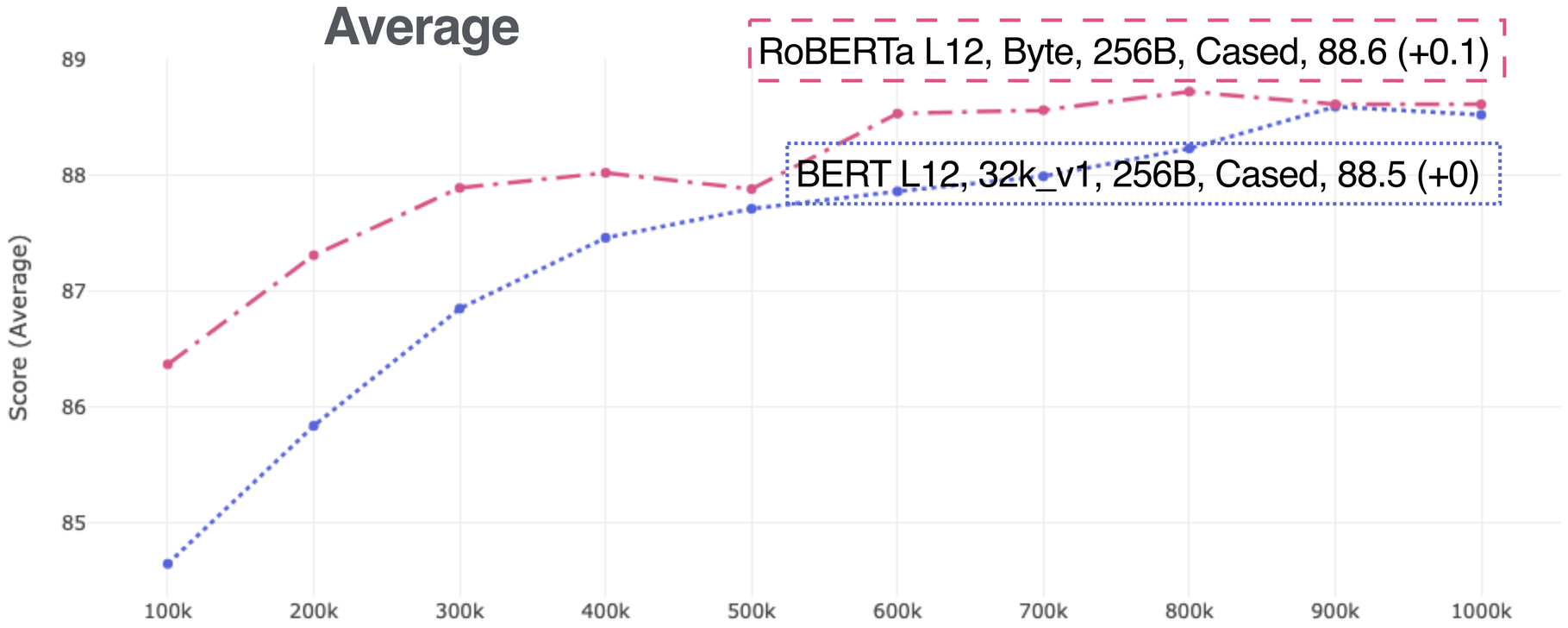




Model: BERT vs RoBERTa

초기에는 RoBERTa가 우세해 보이나, 1M Steps에서는 비슷한 성능

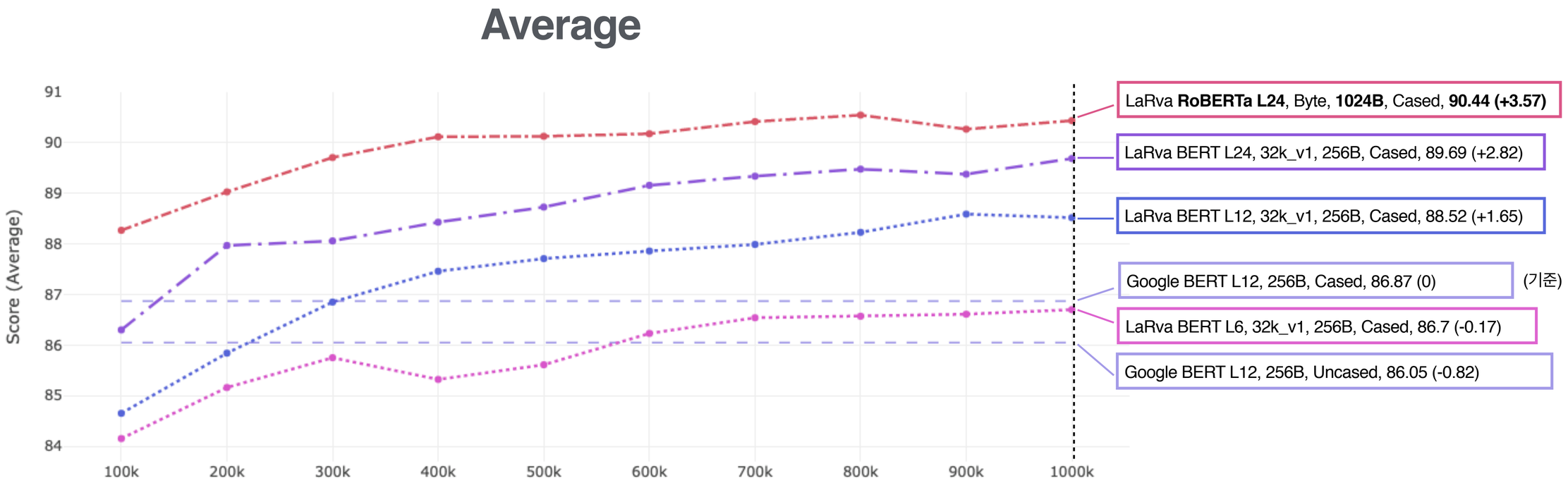
(두 모델 간의 차이점: WordPiece + NSP vs Byte-BPE + without NSP)



모든 좋은 점들 합치면...

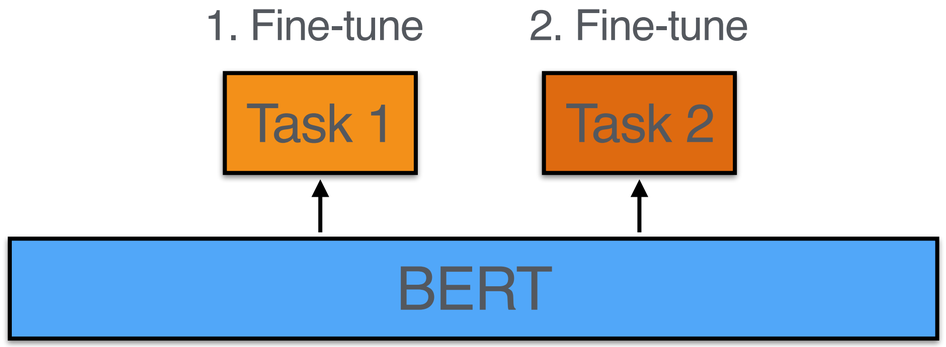
DEVIEW 2019

RoBERTa + Large(L24) + 1024B…

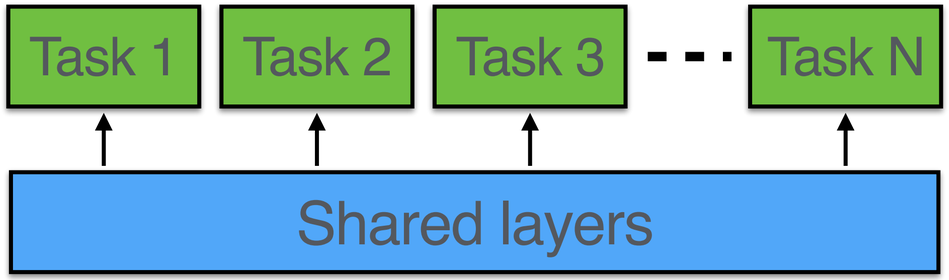


Fine-tuning 관점

- Multi-Task Learning



STILTs on BERT



(Task specific layers)

각 Task에 Fine-tune

MT-DNN

Sentence Encoders on STILTs: Supplementary Training on Intermediate Labeled-data Tasks - 2018, J Phang

Multi-Task Deep Neural Networks for Natural Language Understanding - 2019, X Liu

Multi-Task Learning: MT-DNN

* (- : 성능의 떨어지거나, 거의 비슷한 경우)

	CoLA	MNLI-m/mm	MRPC	QNLI	QQP	RTE	SST	STS-B	WNLI
BERT	61.4	86.5/86.0	89.5	89.9	88.7	70.0	94.0	90.0	56.3
MT-DNN	-	-	**92.1**	**90.3**	-	**83.4**	-	-	-

GLUE Benchmark

	HQA	HQA-long	KorQuAD	MR-2	MR-5	QS	SM	PAWS
LaRva-Kor	95.1	94.3	93.9	90.0	75.5	87.4	94.2	84.9
MT-DNN	**96.9**	-	-	**90.8**	-	**89.2**	**95.0**	-

KLUE Benchmark

* MT-DNN은 CLaF 에 구현되어있습니다.

실험 결과들 간단 정리

- 모든 Task 성능 향상이 도움이 되는 요소:
    - BatchSize ↑, ModelSize ↑, Optimal Vocab
    - Pre-training Methods (Masking Strategy, Full Sentence 등...)
- 각 Task에 의존적인 요소:
    - Tokenizer (Char/Wordpiece), Cased/Uncased
- Pre-training 외에도 Fine-tuning 단 성능 향상 가능
    - Multi-Task Learning - 같은 도메인에서 상대적으로 수가 적은 데이터셋의 성능 향상

4. 서비스는 속도

LaRva for end-user services

LaRva 모델들을 통해, 성능의 향상.

그렇다면, 서비스에 바로 적용하면 되겠다!

이미지: https://namu.wiki/w/리액션%20가이즈

아직은 적용하기 어려운 PLM...

LaRva 모델들을 통해, 성능의 향상.

그렇다면, 서비스에 바로 적용하면 되겠다!

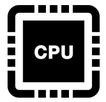
아직 넘어야 할 산이 많아요! :(

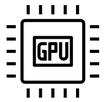
- 처리속도
- 메모리 이슈 (모델 사이즈)
- 비용

이미지: https://namu.wiki/w/리액션%20가이즈

처리속도: Engineering 접근법

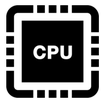
아무런 최적화가 진행되지 않은 BERT-Base (12 layers): (* 1 BatchSize, 128 Length 기준)

- 

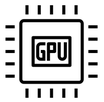
: 약 300 ms, 

: 약 18ms

해결방법

1) cuBERT: 

- Intel MKL-DNN & 멀티 threading
- **52 ms** 까지 감소

2) Nvidia TensorRT: 

- T4 GPU: **8 ms**
- + FP16 (Tensor core) 사용시: 약 **4 ms**
- 더 자세한 실험 결과는 Appendix 참고

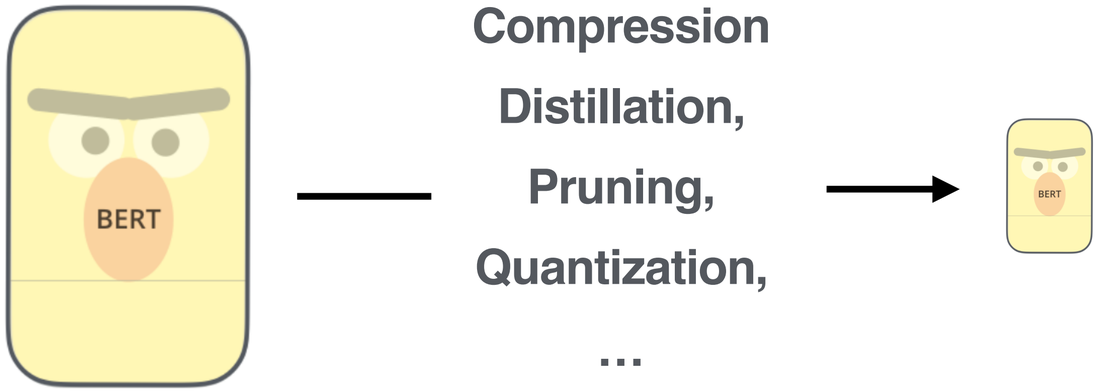
cuBERT: https://github.com/zhihu/cuBERT

TensorRT: https://github.com/NVIDIA/TensorRT

모델의 사이즈가 너무 크다..!

- BERT-Base의 파라미터 수: 110 M, 약 420M
- BERT-Large의 파라미터 수: 340M, 약 1.3G

성능은 차이가 별로 나지 않으면서, 작은 모델이 필요하다! (Future Work)



Special thanks to

DEVIEW 2019

김규완

김민정

김한주

(ㄱㄴㄷ 순)

서민준

성낙호

하정우

Q & A

DEVIEW
2019

Thank You

Appendix: FasterTransformer

DEVIEW 2019

Tesla T4			fp16 latency (ms)				fp16 throughput (sentence/sec)	fp32 latency (ms)				fp32 throughput (sentence/sec)	FP16/FP32 speed up	
BS	model	seq_len	90%	95%	99%	avg	avg	90%	95%	99%	avg	avg	avg	
1	large	512	27.35	27.51	27.93	26.83	37.28	92.68	92.92	93.44	91.91	10.88	0.29	n_head=16
2	large	512	52.93	53.17	53.65	52.03	38.44	193.10	193.64	195.03	191.47	10.45	0.27	
4	large	512	100.68	100.95	101.80	99.53	40.19	366.95	367.66	369.42	363.82	10.99	0.27	
8	large	512	201.80	202.41	203.64	199.57	40.09	746.52	747.04	748.04	741.39	10.79	0.27	
16	large	512	391.46	392.40	397.04	386.95	41.35	1384.70	1385.45	1387.95	1379.39	11.60	0.28	
1	large	128	8.65	8.73	9.33	8.48	117.88	26.28	26.42	26.70	25.48	39.24	0.33	
2	large	128	11.53	11.63	12.29	11.26	177.60	44.54	44.68	45.18	43.73	45.73	0.26	
4	large	128	18.97	19.10	19.56	18.40	217.33	86.96	87.20	87.68	86.12	46.45	0.21	
8	large	128	37.82	38.05	38.50	37.07	214.80	172.24	172.72	173.80	169.72	47.14	0.22	
16	large	128	71.62	72.00	72.89	70.55	226.78	323.60	324.90	325.65	321.17	49.82	0.22	
1	base	512	10.37	10.47	11.07	10.12	98.81	31.64	31.84	32.32	31.07	32.19	0.33	n_head=12
2	base	512	18.91	19.07	20.03	18.41	108.61	61.05	61.30	62.27	59.78	33.46	0.31	
4	base	512	35.29	35.62	36.79	34.77	115.04	116.71	116.94	117.83	115.29	34.70	0.30	
8	base	512	68.08	68.56	69.56	67.17	119.10	221.87	222.61	225.14	219.76	36.40	0.31	
16	base	512	129.38	130.54	132.71	128.16	124.85	450.46	451.83	455.76	445.33	35.93	0.29	
1	base	128	4.12	4.19	4.45	3.95	253.13	8.44	8.50	8.78	8.28	120.73	0.48	
2	base	128	4.88	4.94	5.40	4.79	417.49	15.36	15.47	15.87	15.06	132.81	0.32	
4	base	128	7.17	7.24	7.77	7.12	561.54	25.98	26.13	26.54	25.46	157.08	0.28	
8	base	128	12.72	12.81	13.15	12.57	636.50	48.06	48.31	48.71	47.32	169.06	0.27	
16	base	128	24.13	24.27	27.80	23.79	672.42	96.55	97.06	99.45	95.05	168.33	0.25	

* no use_xla

이미지: https://namu.wiki/w/리액션%20가이즈