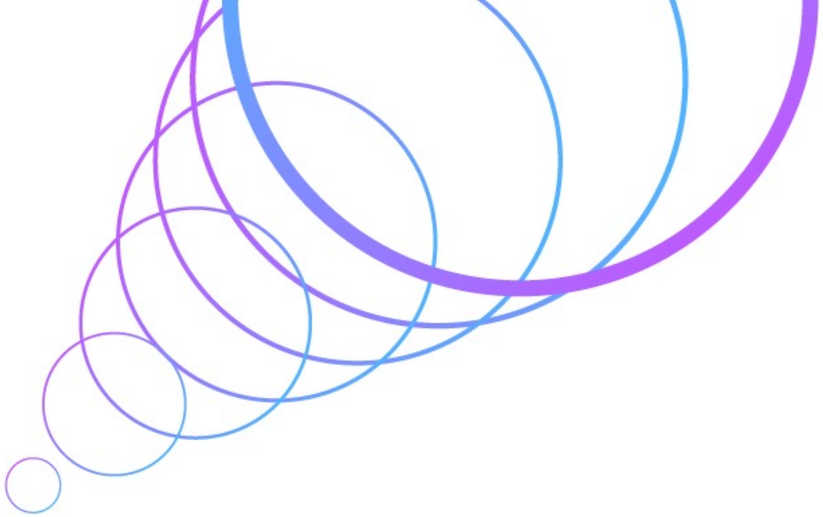
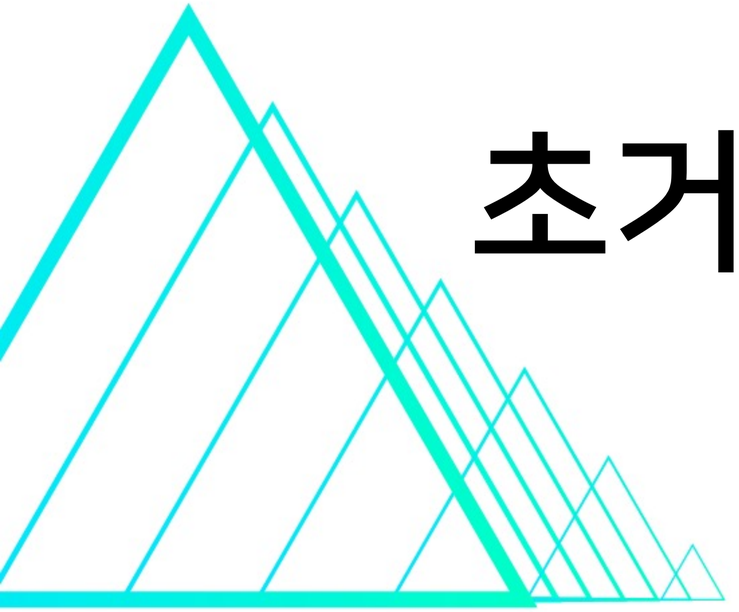




초거대 AI ‘HyperCLOVA’ 서빙기

김민섭 이성재 NAVER / Next ML Inference TF

CONTENTS

1. HyperCLOVA Overview

1.1 HyperCLOVA란?

HyperCLOVA: NAVER의 초거대 언어 모델

- 텍스트 생성
- 단어 간 연관도 추출로 추천시스템에도 활용 가능
- 현재 클로바 스피커, 챗봇 등 네이버의 다양한 AI 서비스에 활용되고 있음
- 더 좋은 성능을 위해 모델 크기가 점점 커지고 있는 추세임

안녕하세요

반갑습니다…

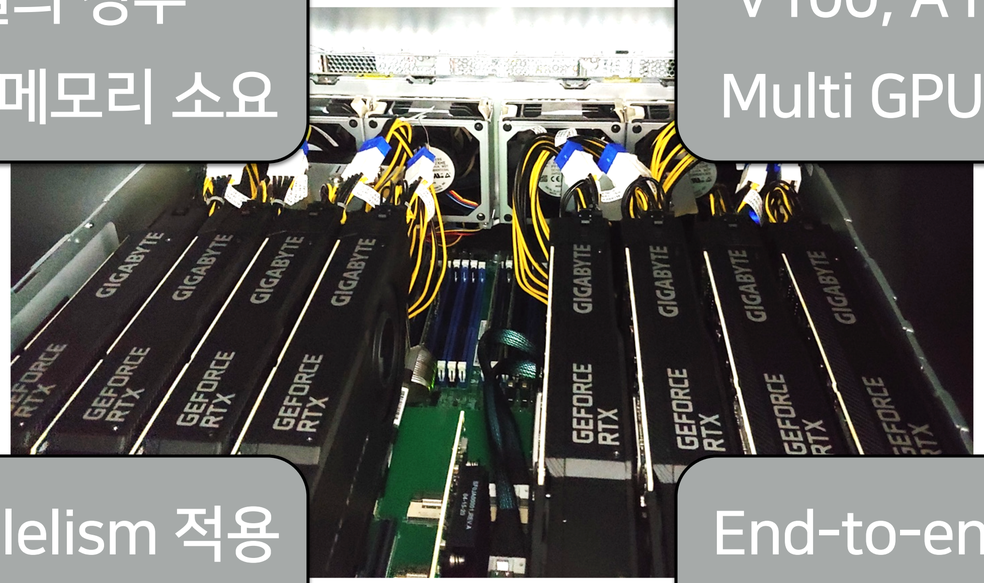
HyperCLOVA
서빙 로직

1.2 초거대 모델 서빙의 어려운 점

- 규모와 호환성: 아직 거대 모델을 안정적으로 돌릴 수 있는 프레임워크가 부족
- 시간: 많은 연산량 → 긴 지연시간
- 비용: 비싼 GPU 자원

큰 모델의 경우
160GB의 메모리 소요

V100, A100 GPUs
Multi GPU 통신 필요

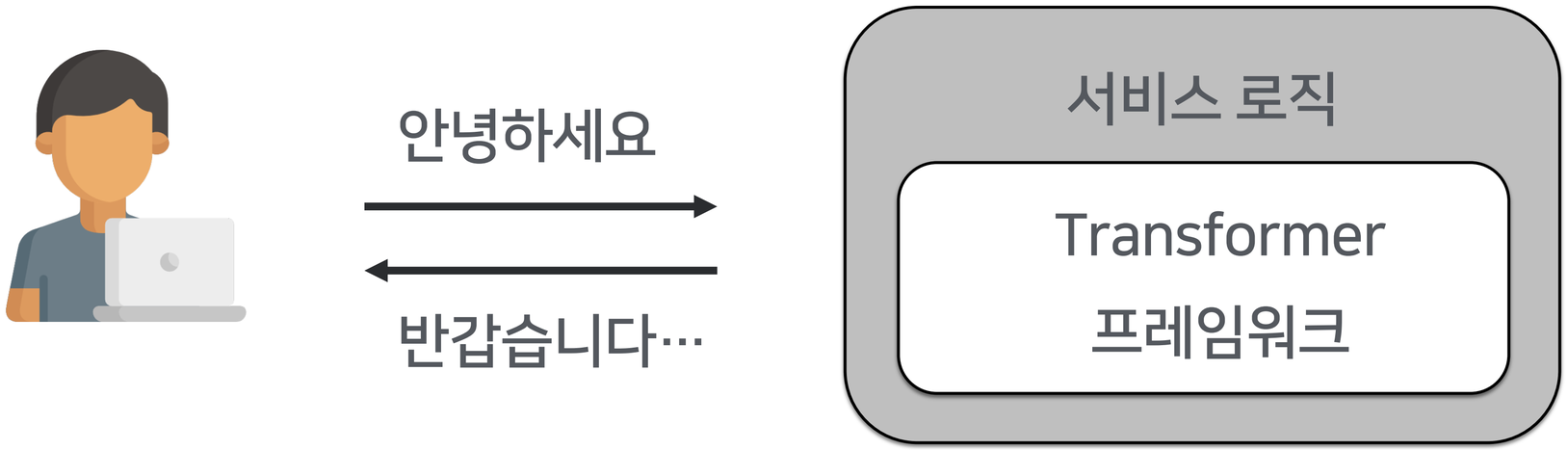


각종 parallelism 적용
가능한 framework

End-to-end latency
단축 필요

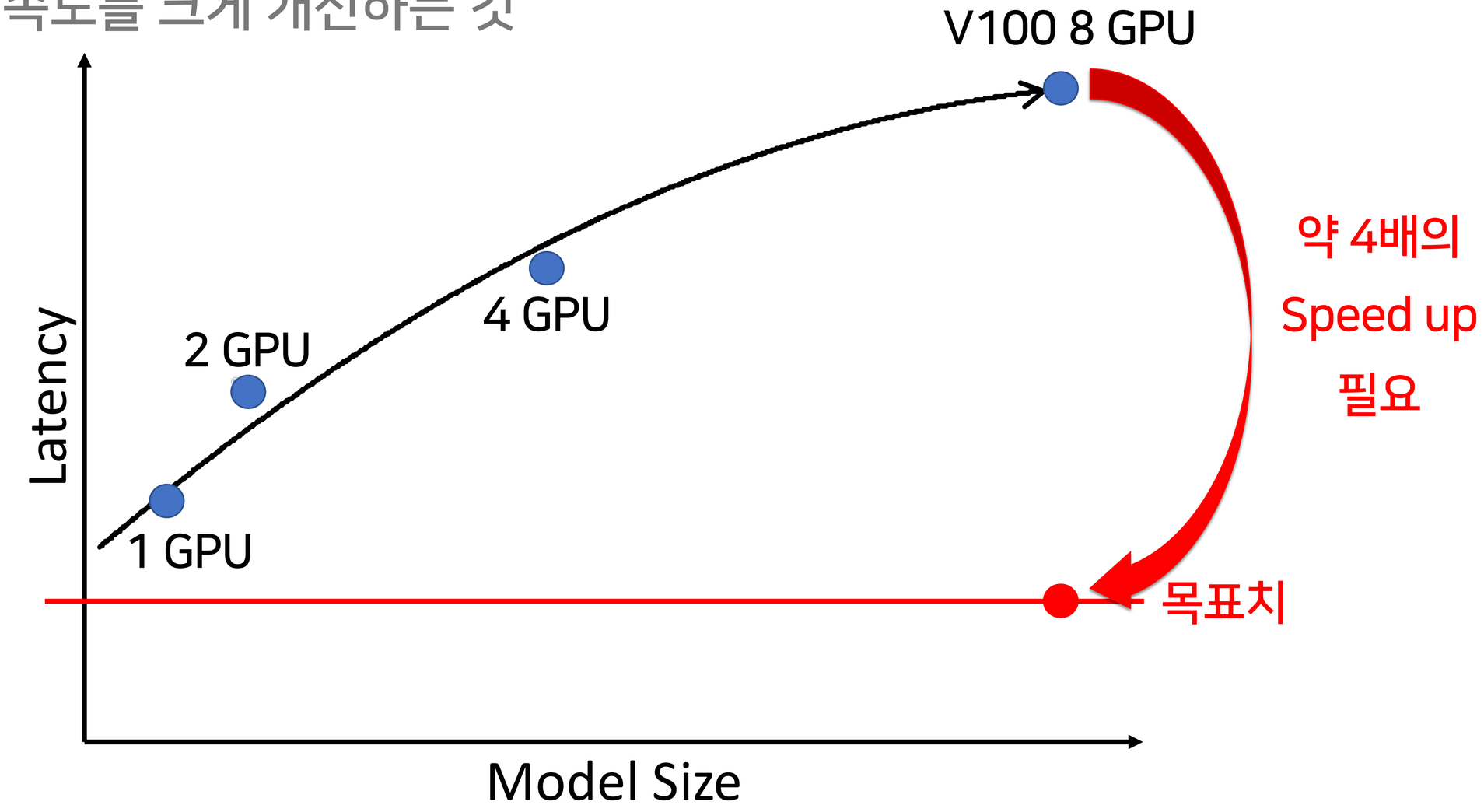
1.3 개발 시작 시기의 상황

- 내부 프레임워크에서 transformer 연산을 통해 텍스트 생성
- 서비스 로직은 transformer 연산 프레임워크 바깥에서 요청을 받아 적절히 응답
- 학습에서 사용한 프레임워크를 서비스에 그대로 사용



1.4 개발 목표

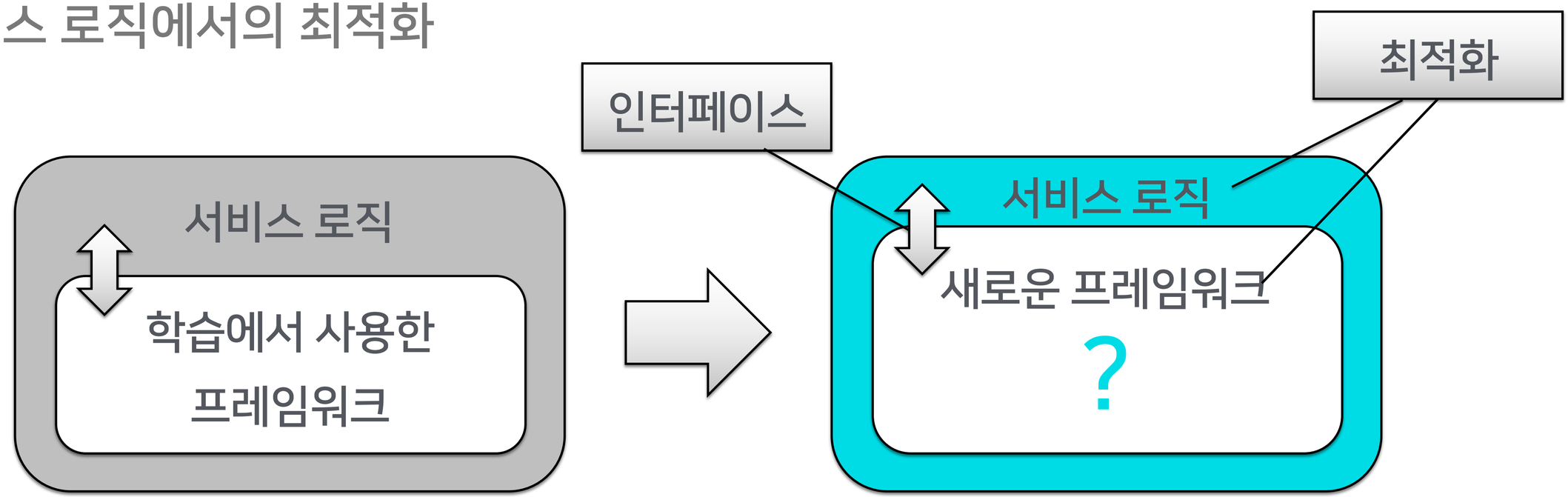
- Inference 속도를 크게 개선하는 것



1.4 개발 목표

How?

- Transformer 연산 최적화. 새로운 프레임워크 적용 시도
- 프레임워크와 서비스 로직 사이의 인터페이스도 변경 필요
- 서비스 로직에서의 최적화

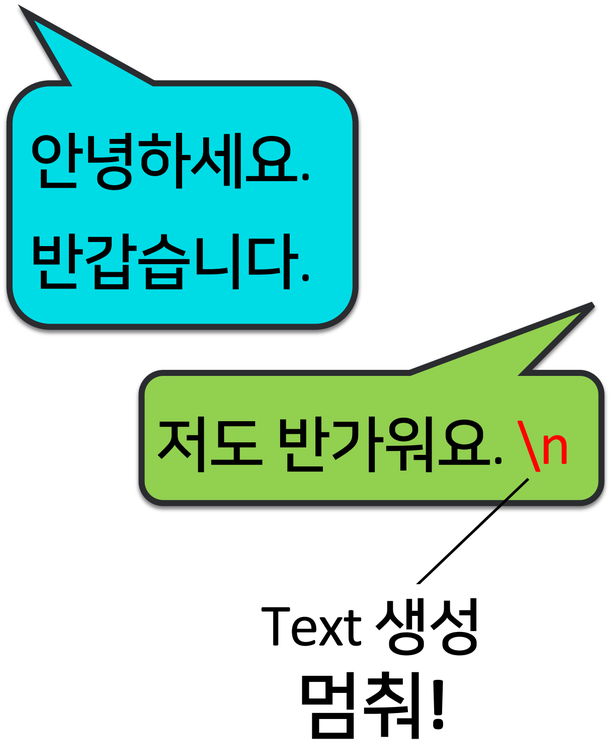


1.4 개발 목표

DEVIEW 2021

- 서비스에 필요한 추가 기능들 구현

- Early Stop



- Semantic Search

- Prompt Tuning

최신 연구 내용 적용

The power of scale for parameter-efficient prompt tuning
B Lester, R Al-Rfou, N Constant - arXiv preprint arXiv:2104.08691, 2021 - arxiv.org
In this work, we explore" **prompt tuning**", a simple yet effective mechanism for learning" soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation …
☆ 99 31회 인용 관련 학술자료 전체 2개의 버전 »

Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification
S Hu, N Ding, H Wang, Z Liu, J Li, M Sun - arXiv preprint arXiv:2108.02035, 2021 - arxiv.org
Tuning pre-trained language models (PLMs) with task-specific prompts has been a promising approach for text classification. Particularly, previous studies suggest that **prompt-tuning** has remarkable superiority in the low-data scenario over the generic fine-tuning …
☆ 99 6회 인용 전체 3개의 버전 »

PTR: Prompt Tuning with Rules for Text Classification
X Han, W Zhao, N Ding, Z Liu, M Sun - arXiv preprint arXiv:2105.11259, 2021 - arxiv.org
Fine-tuned pre-trained language models (PLMs) have achieved awesome performance on almost all NLP tasks. By using additional prompts to fine-tune PLMs, we can further stimulate the rich knowledge distributed in PLMs to better serve downstream task. **Prompt tuning** has …
☆ 99 13회 인용 관련 학술자료 전체 2개의 버전 »

2.Transformer 프레임워크 변경

2.1 기존 프레임워크의 한계

- 학습에 사용한 프레임워크는 inference 연산에 최적화 되지 않음
- 실제 서비스에 적용하기 위해서는 약 4배 정도의 속도 향상이 필요

- FasterTransformer: NVIDIA의 오픈소스 GPT inference 프로젝트
- 학습에 사용한 프레임워크로 생성된 체크포인트(weight, bias 등 포함)을 쉽게 변형하여 FasterTransformer에서 사용할 수 있음

2.2 FasterTransformer의 가능성

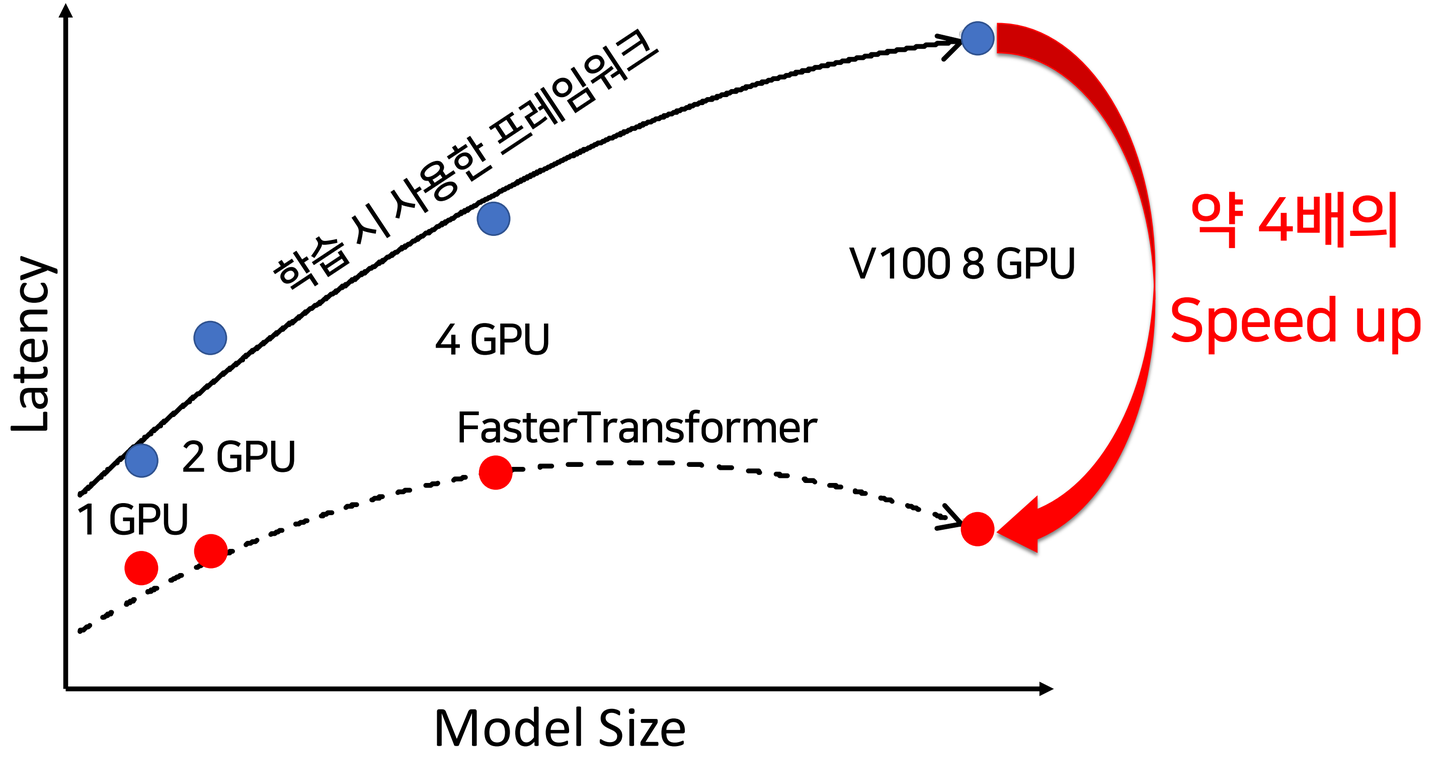
DEVIEW 2021

FasterTransformer가 빠를 수 있는 이유

- Pytorch가 아닌 C++과 CUDA native kernel로 작성되어 있음
- Kernel fusion 등 CUDA 커널 레벨 최적화 기법 적용
- Collective communication의 오버헤드가 더 적도록 설계
- Faster Transformer: CUDA를 이용한 BERT inference 최적화 (DEVIEW2019)

2.3 FasterTransformer를 통한 속도 향상

- 모델 크기에 따라 차이가 있지만, 최종적으로 학습에 사용한 프레임워크 대비 최대 4배 속도 향상
- 프레임워크 변경만으로 목표 속도를 달성할 수 있었음



2.4 정합성 문제들

- 그러나 올바르지 못한 결과가 자주 발생(정합성 문제)
- FasterTransformer v4.0 기준
- 오른쪽 예시와 같이 훼손된 텍스트가 생성되는 현상

- 입력

안녕하세요.반갑습니다.

- 올바른 출력 예시

안녕하세요. 저는 네이버 클로바입니다.

- 완전히 잘못된 출력

RRRRRRRRRRRRRRRRRR

- 일부 말이 안되는 단어 생성

안녕하세요. 저는 ,,,,,,,,,,,,,,,,,,;;--

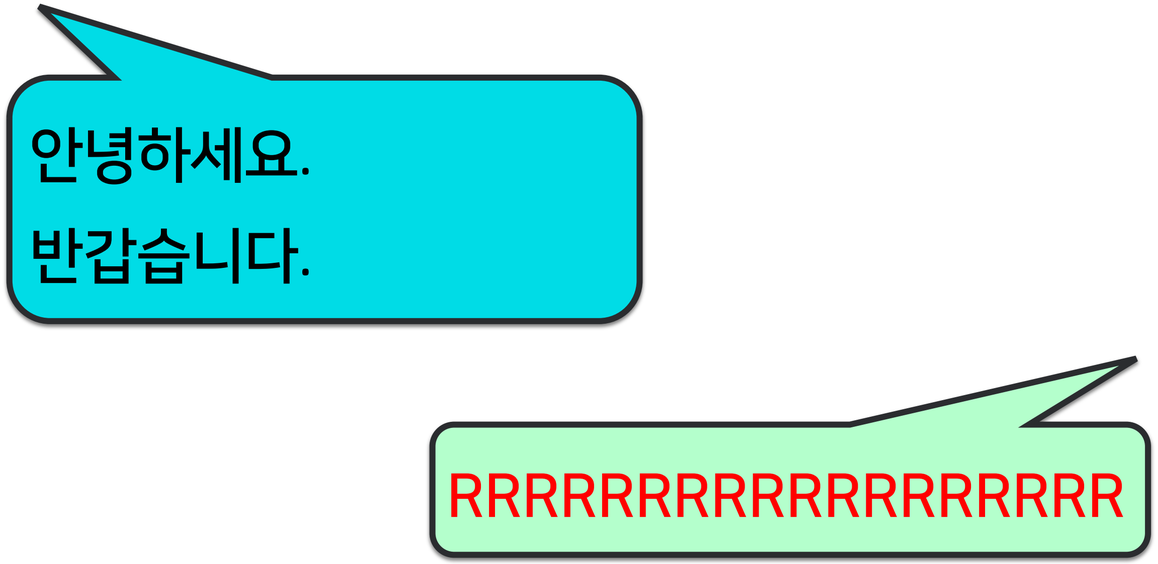
- 잘못된 조사

안녕하세요. 저는는 네이버 클로바입입니다.

2.4.1 체크포인트 변환 시의 문제

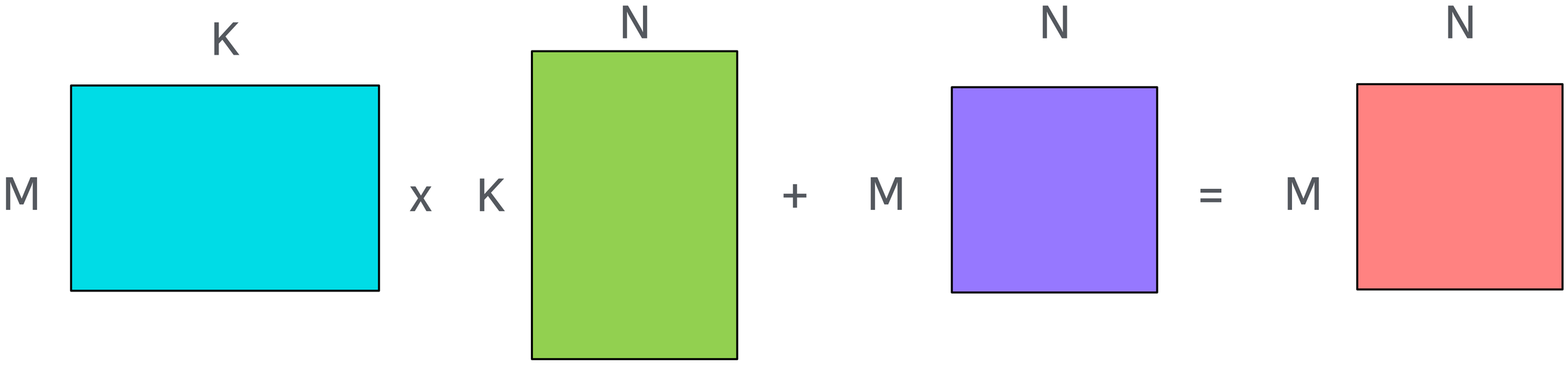
- 기존 프레임워크에서 생산한 모델 체크포인트를 FasterTransformer에 맞춰 변환해야 함
- 체크포인트 변환 시 indexing 오류로 인해 완전히 잘못된 단어들이 생성됨

현상 예시)



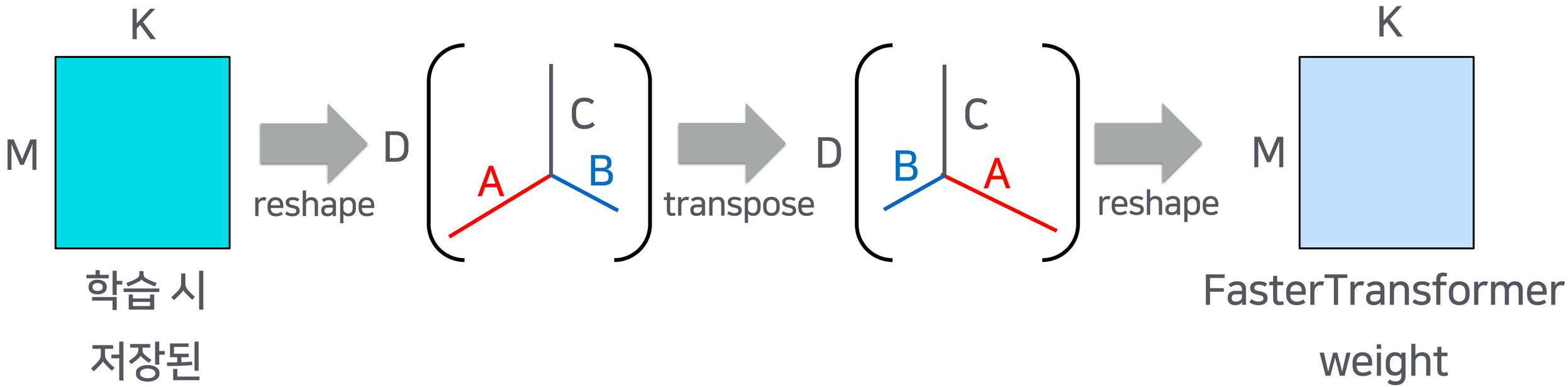
2.4.1 체크포인트 변환 시의 문제

- 각 layer에서 실행되는 weight와 bias 이용한 연산 과정
- Weight와 bias의 dimension은 맞기 때문에 runtime error 없이 진행됨
- Weight와 bias 내부 값들의 위치가 바뀌어야 함



2.4.1 체크포인트 변환 시의 문제

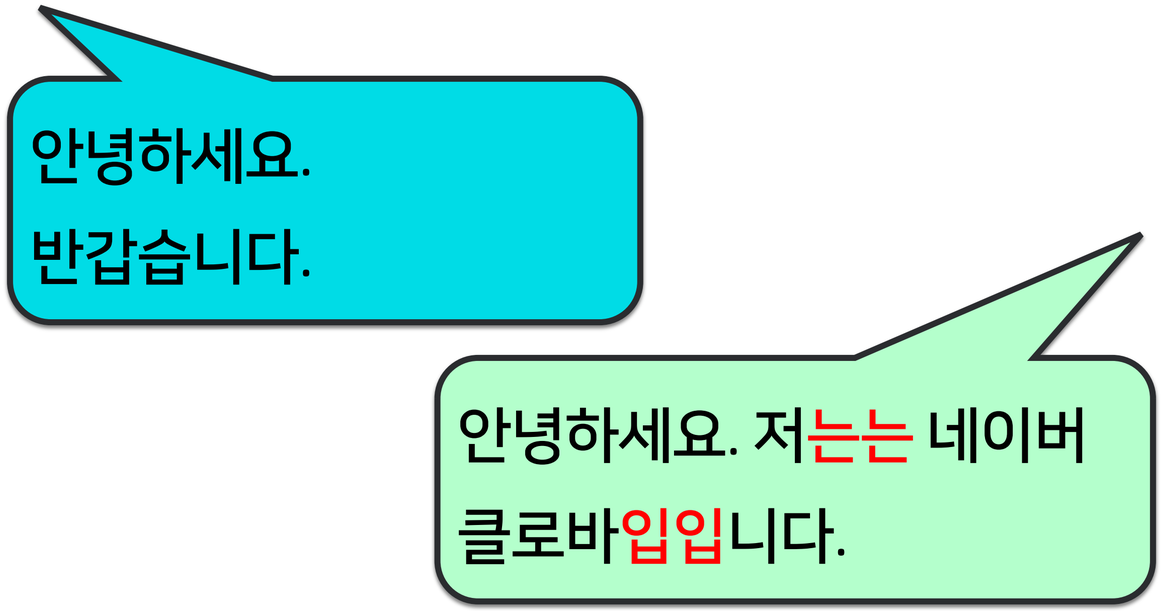
- 체크포인트를 적절히 transpose하여 문제 해결
- 예시) Weight 변환 과정



2.4.2 Race Condition 문제

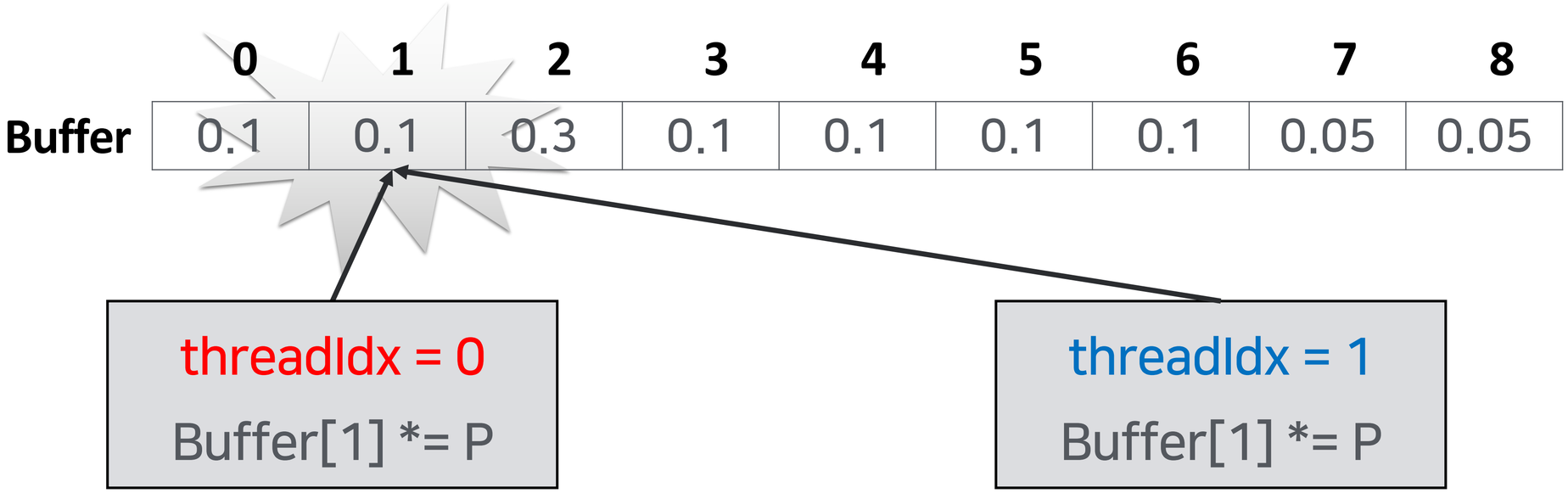
- 간헐적으로 이상한 조사가 생성되는 케이스 발생
- CUDA kernel 내부의 race condition이 원인

현상 예시)



2.4.2 Race Condition 문제

- GPU kernel 내부의 스레드들이 서로 같은 위치의 데이터 버퍼의 값을 갱신하는 과정에서 race condition 발생함

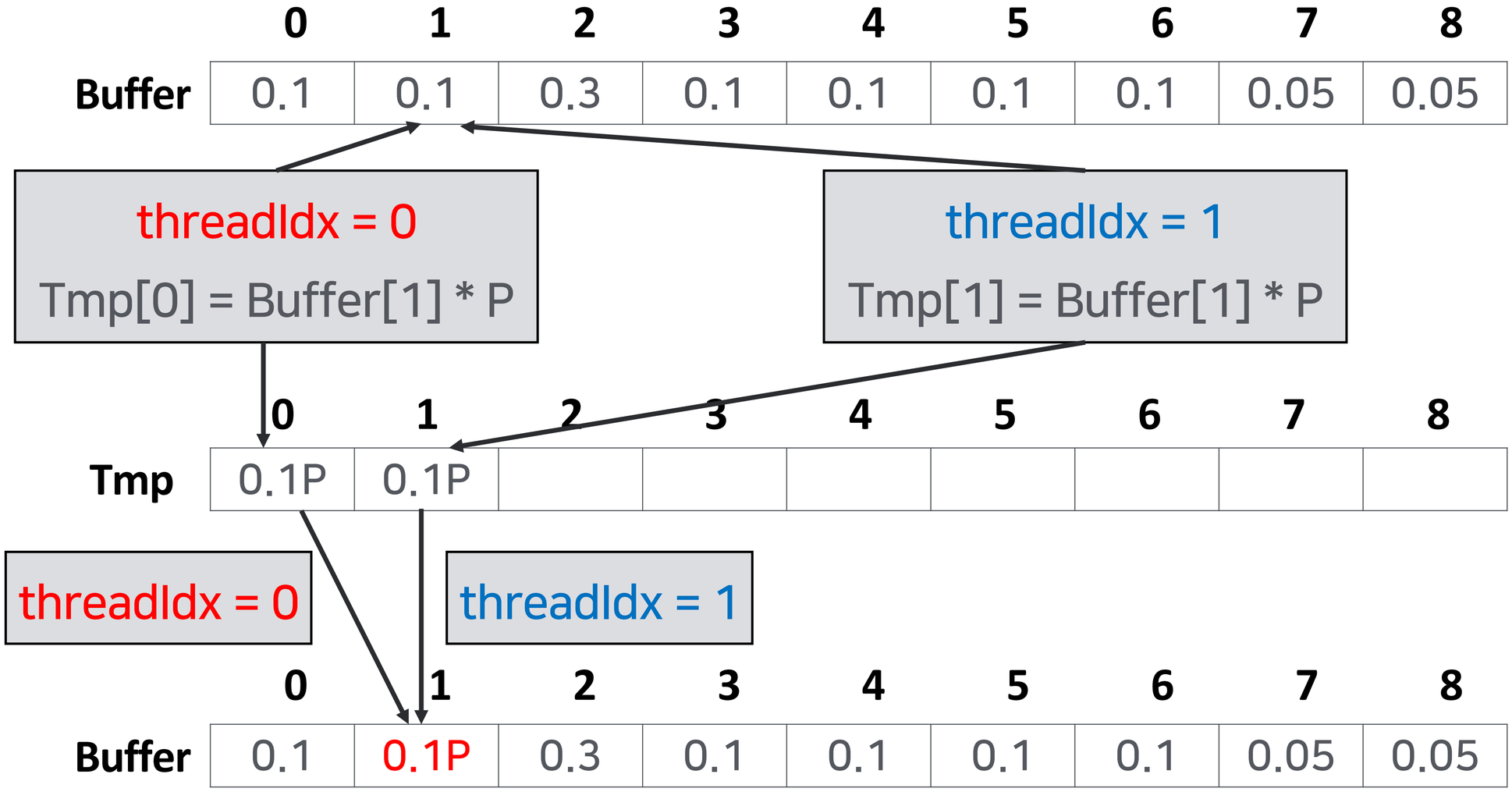


의도한 결과 : Buffer[1] = 0.1 * P,

Race condition으로 인해 발생할 수 있는 결과 : Buffer[1] = 0.1 * P 또는 0.1 * P^2

2.4.2 Race Condition 문제

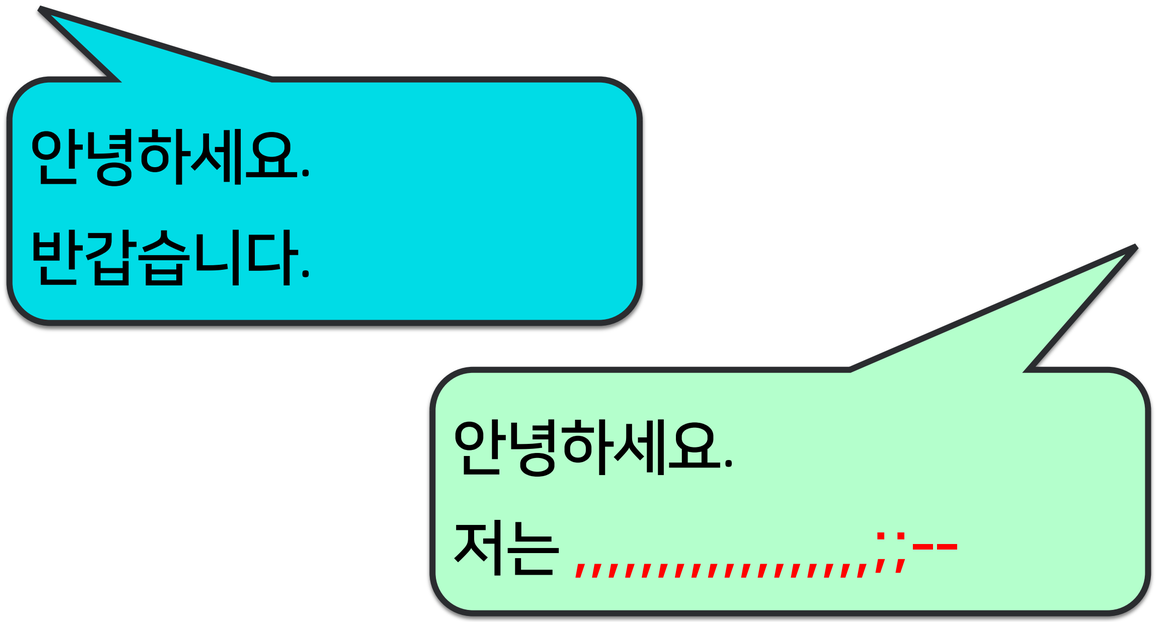
- 수정 후의 kernel 연산 과정)



2.4.3 FP16 적용의 어려움 - Overflow

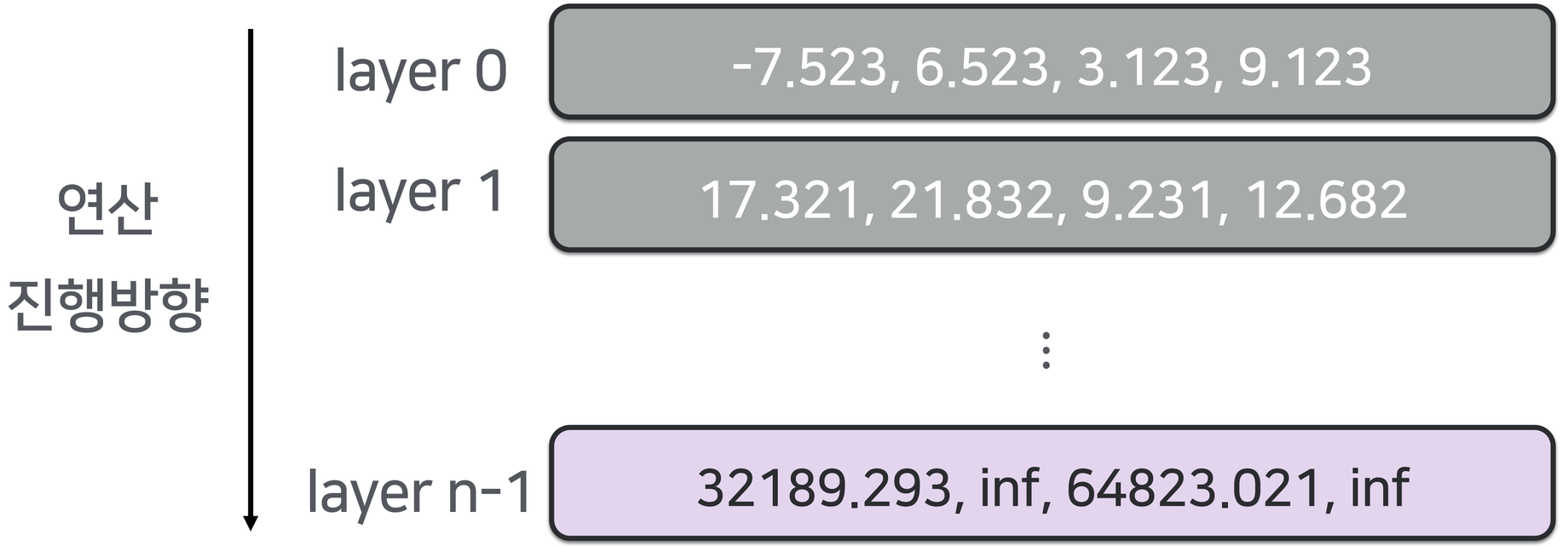
- HyperCLOVA 케이스에서 FP32 대비 FP16 연산이 약 3-4배 빠름
- FP16 데이터타입으로 inference 진행
- Overflow로 인해 일부 말이 안되는 단어들이 생성됨

현상 예시)



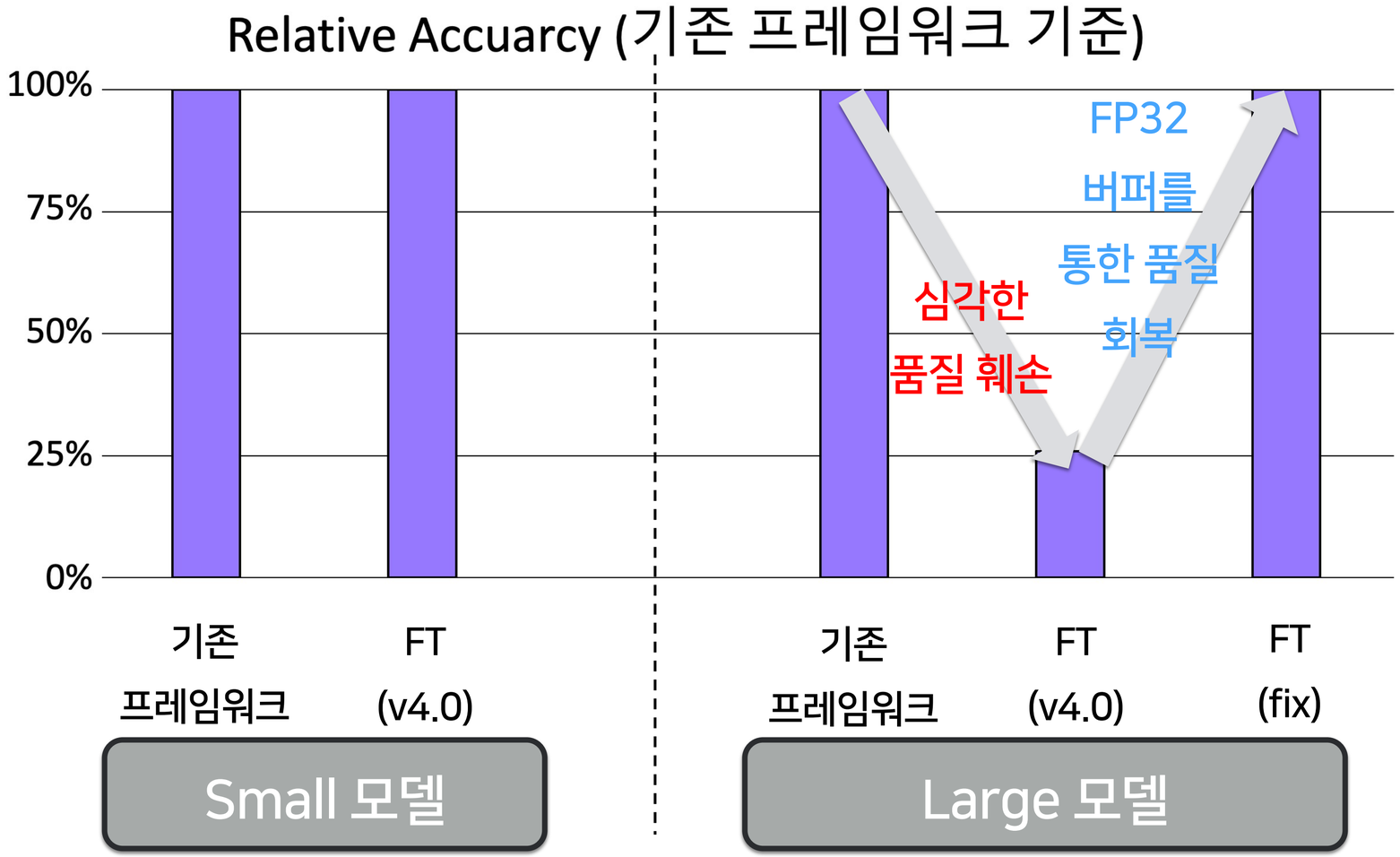
2.4.3 FP16 적용의 어려움 - Overflow

- 뒤쪽 레이어로 갈 수록 GEMM (matrix multiplication) 연산 결과값이 커지는 경향이 있음
- FP16의 최댓값(65504) 보다 클 경우 overflow 발생
- inf, NaN 값으로 인해 특정 시점 이후 잘못된 단어만 생성됨



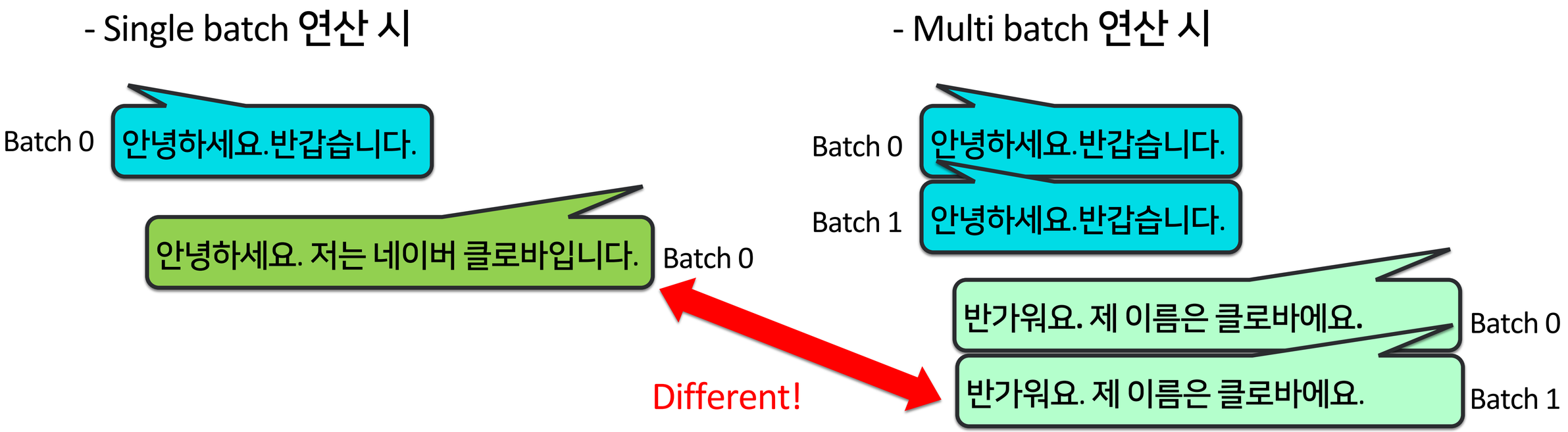
2.4.3 FP16 적용의 어려움 - Overflow

- Overflow 가능성이 높은 GEMM 연산은 FP32 데이터 타입으로 진행하도록 수정



2.4.4 FP16 적용의 어려움 - Swamping

- 동일한 입력과 greedy 옵션 (random한 연산 부분 배제)을 사용할 경우 배치 크기와 상관없이 동일한 출력이 기대됨
- 그러나 예상과 달리 배치 크기에 따라 출력이 달라짐

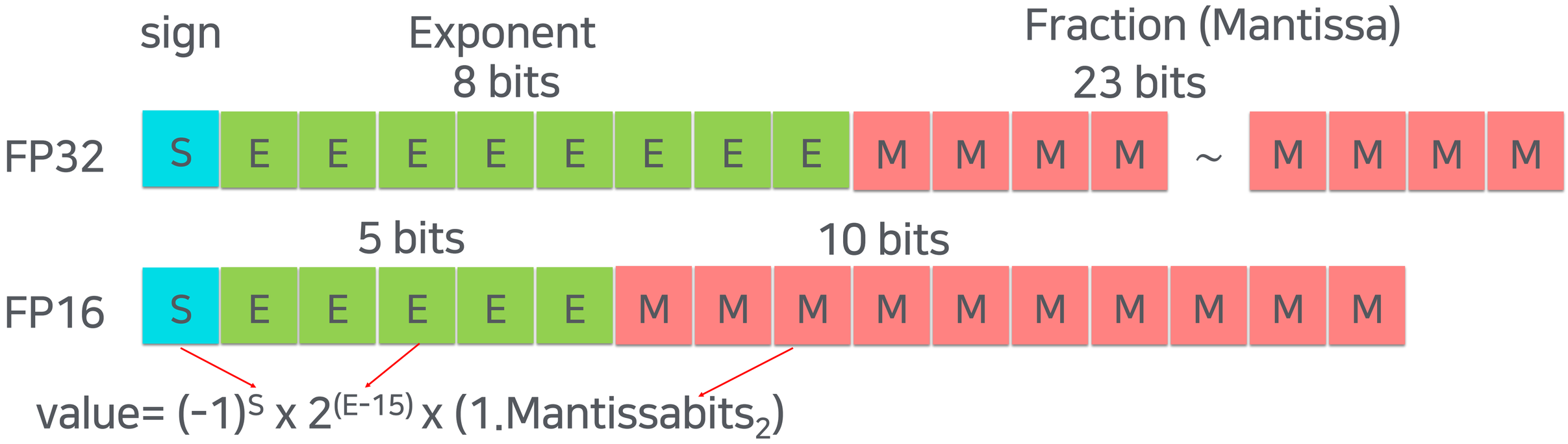


2.4.4 FP16 적용의 어려움 - Swamping

- GEMM 연산 과정 중 배치 사이즈에 따라 결과가 달라짐을 확인함
- 배치 크기에 따라 input matrix의 크기가 달라질 때, GEMM 연산 kernel 내부에서 summation 순서가 바뀔 수 있음에 유의함

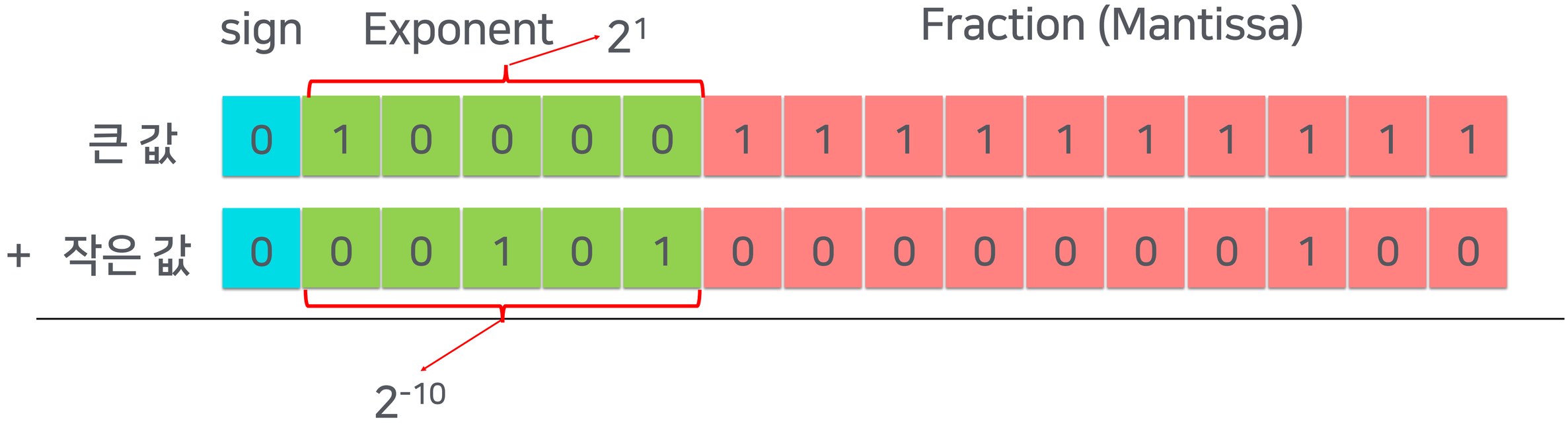
2.4.4 FP16 적용의 어려움 - Swamping

- Summation 순서가 연산 결과에 영향을 미칠 수 있는 이유?
- FP16 타입의 경우 FP32 데이터 타입 대비 fraction (mantissa) bits가 매우 적음
- 큰 값과 작은 값을 더할 때, 작은 값의 데이터가 유실 될 수 있음
- Summation 시 어떤 값들이 먼저 연산 되느냐에 따라 유실되는 데이터가 다름



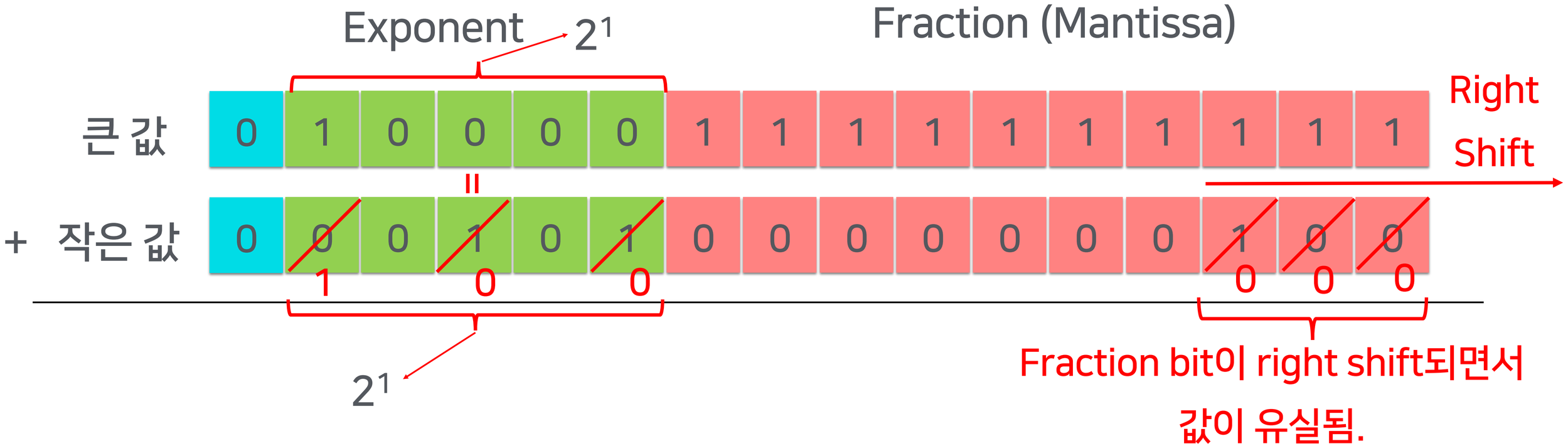
2.4.4 FP16 적용의 어려움 - Swamping

- 큰 값과 작은 값을 더할 때, exponent bits를 맞추다 보니 작은 값의 fraction bit 데이터가 유실될 수 있음
- 이러한 현상이 중첩되면 배치 크기에 따라 연산 결과가 달라질 수 있고, 결과적으로 다른 단어가 선택됨



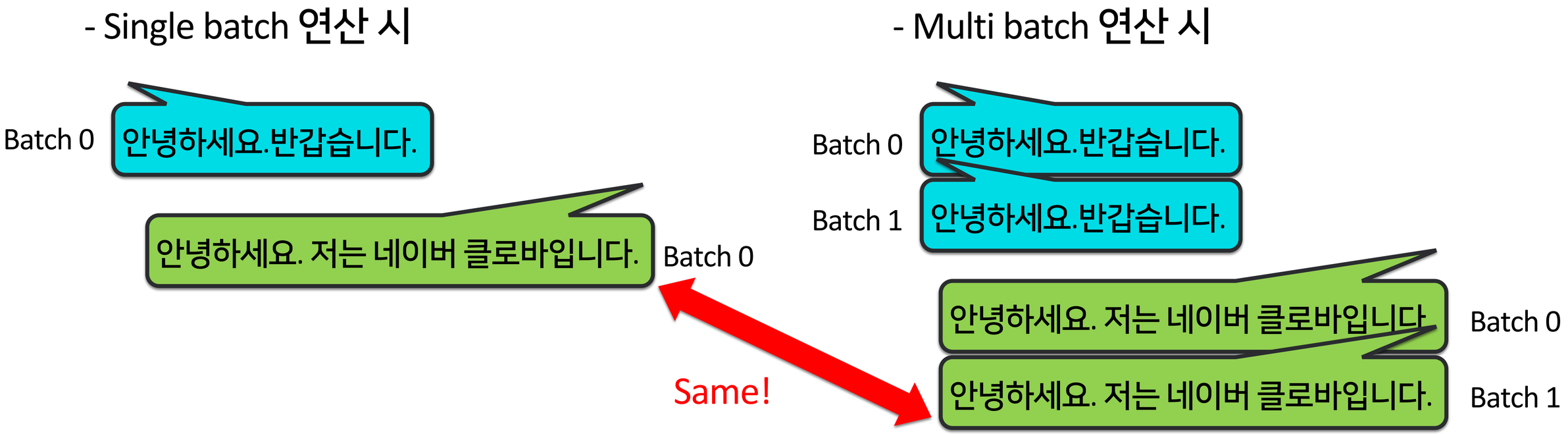
2.4.4 FP16 적용의 어려움 - Swamping

- 큰 값과 작은 값을 더할 때, exponent bits를 맞추다 보니 작은 값의 fraction bit 데이터가 유실될 수 있음
- 이러한 현상이 중첩되면 배치 크기에 따라 연산 결과가 달라질 수 있고, 결과적으로 다른 단어가 선택됨



2.4.4 FP16 적용의 어려움 - Swamping

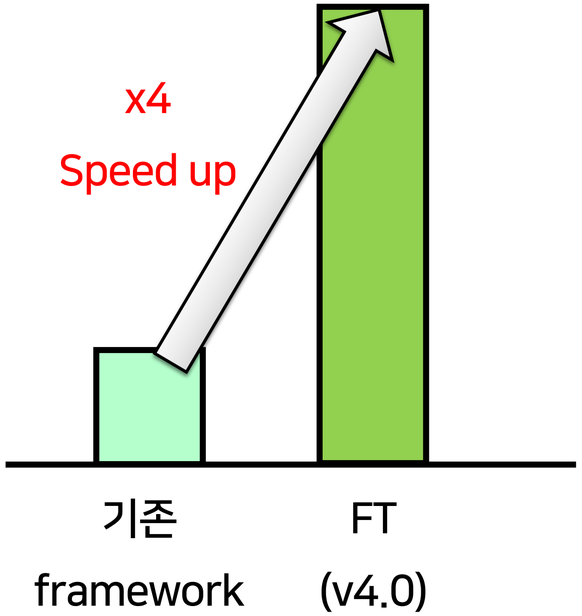
- 데이터타입은 그대로 FP16을 사용하되, GEMM kernel의 중간 summation buffer를 FP32로 설정하여 해당 현상을 완화했음
- FP32 타입의 summation buffer를 사용할 시, 충분한 fraction(mantissa) bit을 통해 데이터 유실 가능성을 줄일 수 있음



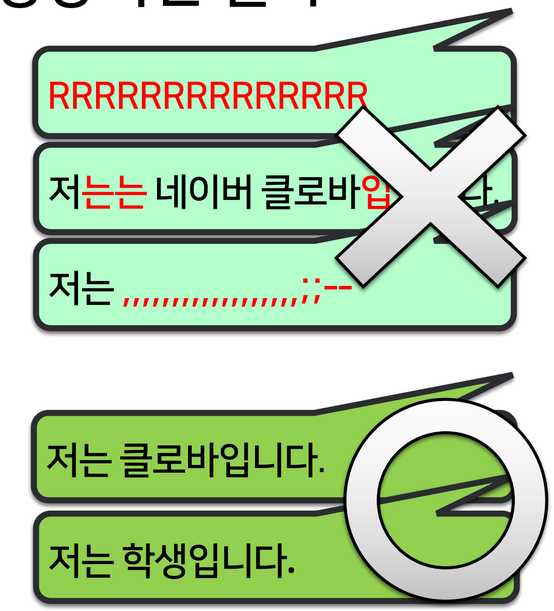
2.5 결과

- FasterTransformer로 프레임워크를 변경하여 HyperCLOVA의 경우 최대 4배의 속도 향상을 이뤘고, 정합성 이슈를 해결하여 실제 서비스에 사용할 수 있게 되었음

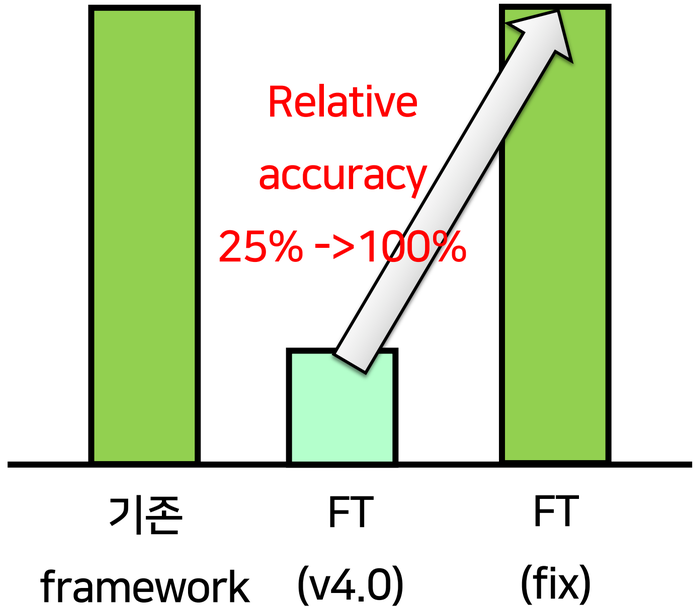
- 속도 증가



- 정상적인 결과



- Text 생성 품질 회복



2.5 결과

- 위의 정합성 문제를 해결해 FasterTransformer v5.0 beta에 기여

Release notes

Changelog

August 2021

- **Release the FasterTransformer 5.0**
  - Refactor the repo and codes
  - And special thanks to NAVER Corp. for contributing a lot to this version, as listed below.
    - Bugs fix
      - Fix error that occurs when batch_size is less than max_batch_size for gpt pytorch wrapper.
      - Fix memory leak that occurs every forward because of reused allocator.
      - Fix race condition that occurs in repetition penalty kernel.
    - Enhancement
      - Add random seed setting.
      - Fix GEMM buffer overflow on FP16 of GPT.
      - Change to invalidate finished buffer for every completion.
      - Introduce stop_before for early stop.

https://github.com/NVIDIA/FasterTransformer/tree/dev/v5.0_beta

3.서빙 백엔드 만들기

3.1 개발 배경

요구 사항

1. FasterTransformer 호환 X
2. 프로세스 관리 O
3. 사내 플랫폼 이용 O

기존의 서빙 백엔드?

3.1 개발 배경

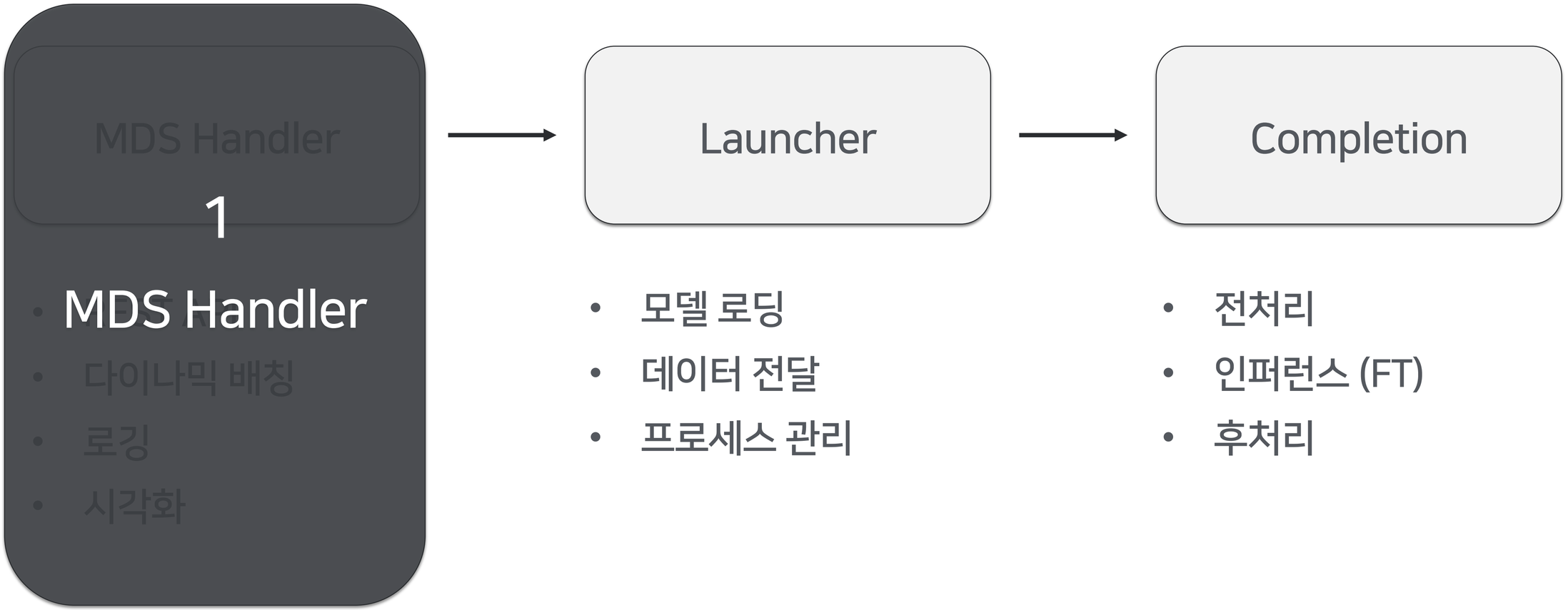
DEVIEW 2021

기존 프레임워크와 주요 FasterTransformer의 차이점

	기존 프레임워크	FasterTransformer
1.Transformer 연산 구조 2. API 접근 방법	• PyTorch 구현 • Python module로 제공	• CUDA native kernel 구현 • Shared library 형태로 결과만 API로 제공

⇒ FTserve: 기존 서빙 앱 구조를 가져 가면서 **서비스 로직**은 FasterTransformer의 **CUDA native kernel**에 맞게 변경하고, **서비스 로직**의 구현 위치를 **python에서 C++로 이동**시킨 서빙 앱

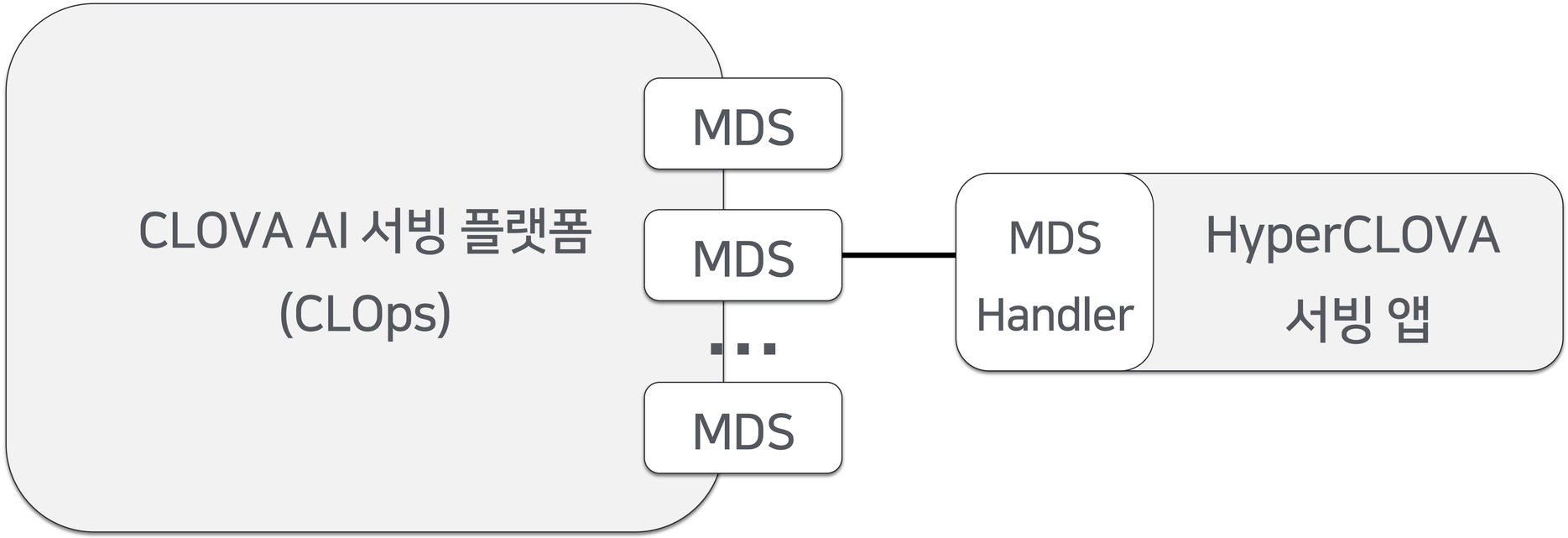
3.2 FTserve: 새로운 HyperCLOVA 서빙 앱



3.3 FTserve: MDS Handler

MDS(Modern Deep learning Serving)

MDS is python framework package for serving diverse AI models

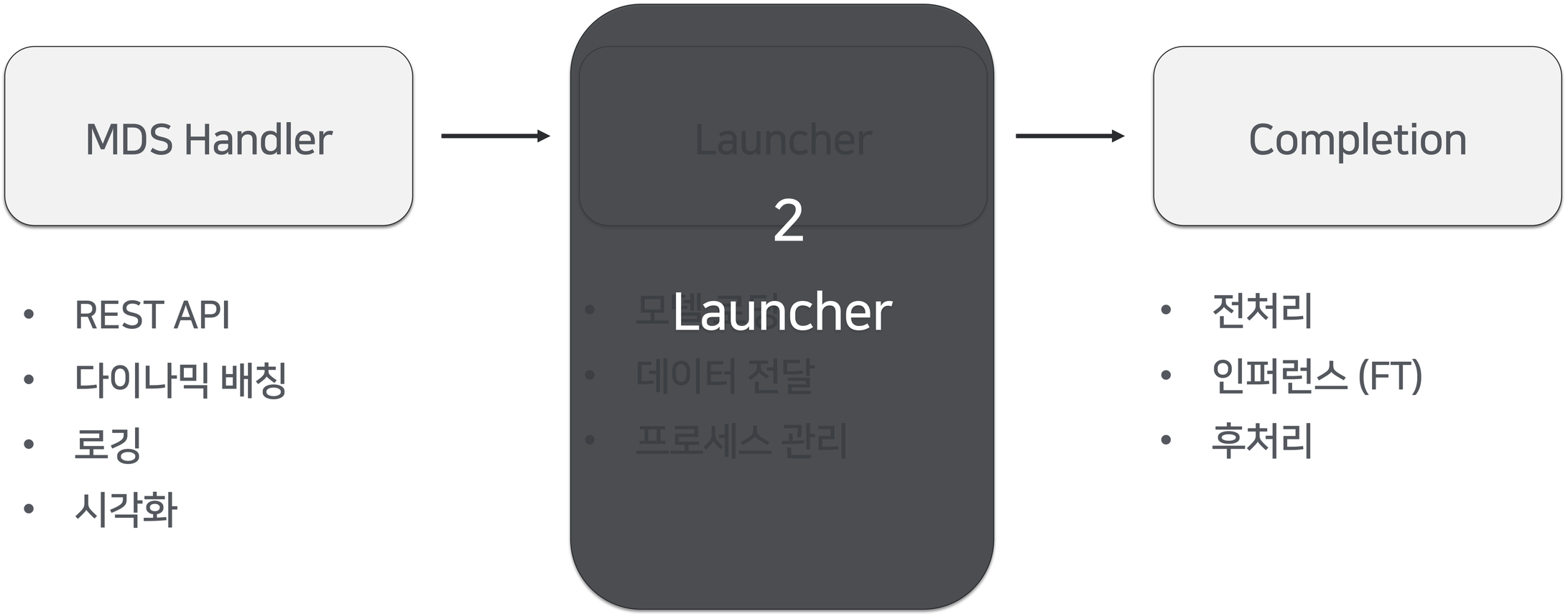


참조)

밑바닥부터 만드는 인공지능 서빙 플랫폼(deview 2020) - https://tv.naver.com/v/16968271

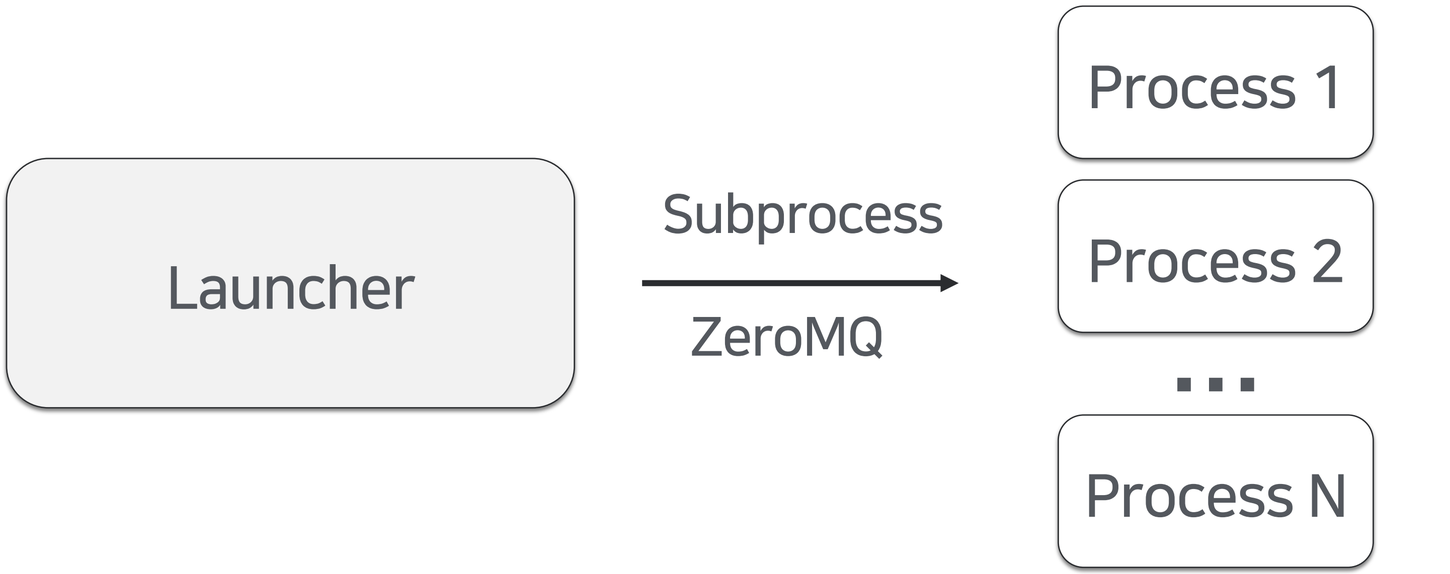
CLOps: 넌 모델링만해, 난 서빙할테니(deview 2021)

3.4 FTserve: Launcher



3.4 FTserve: Launcher

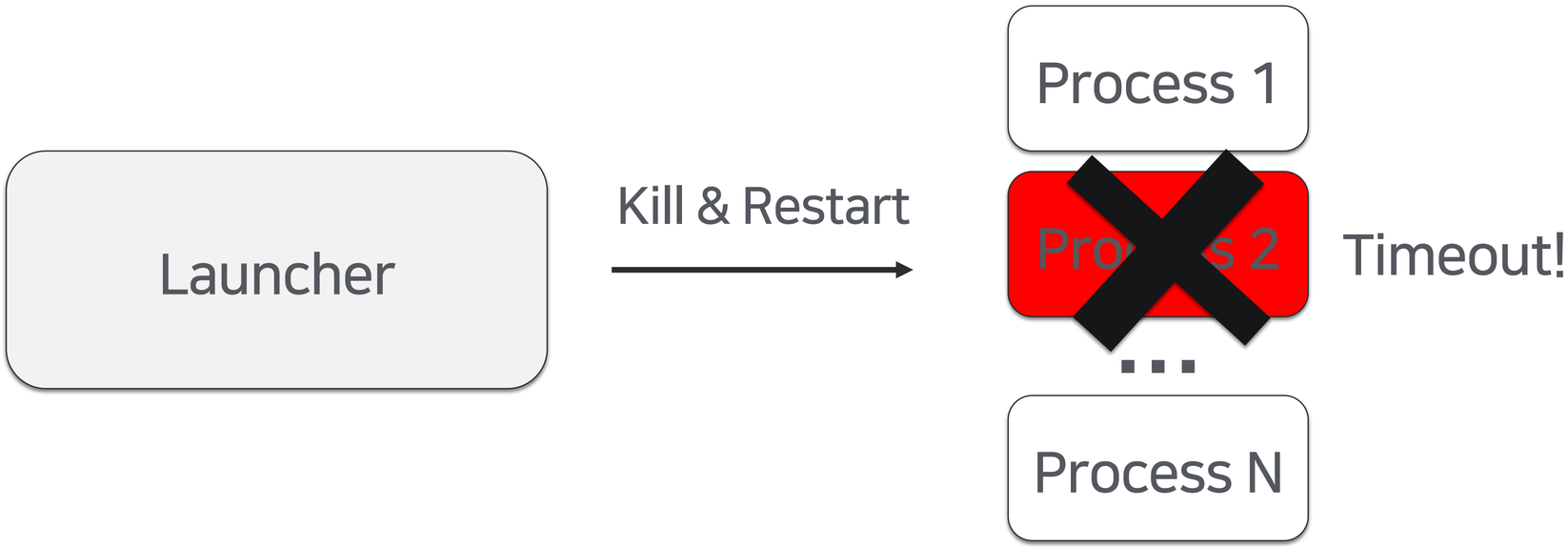
Launcher : 모델 로딩 및 데이터 전달



Model Parallelism

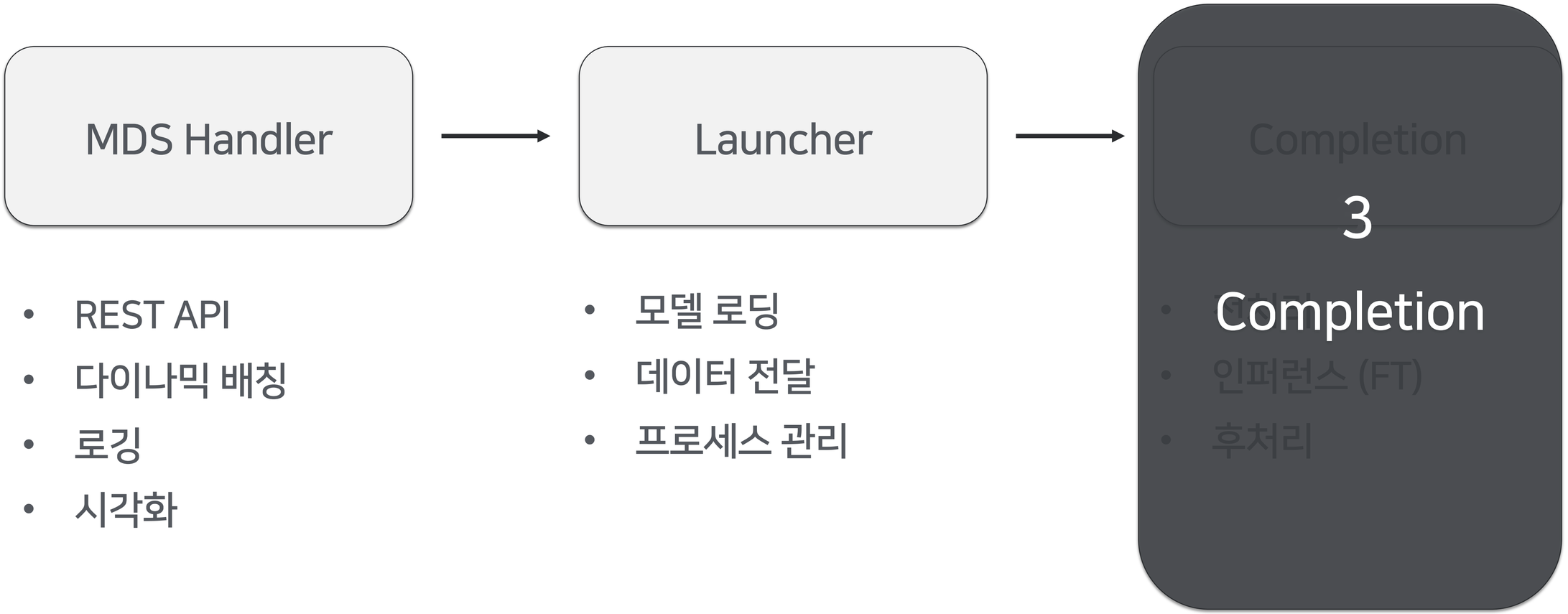
3.4 FTserve: Launcher

Launcher : 프로세스 관리



Model Parallelism

3.5 FTserve: Completion



- REST API
- 다이나믹 배칭
- 로깅
- 시각화

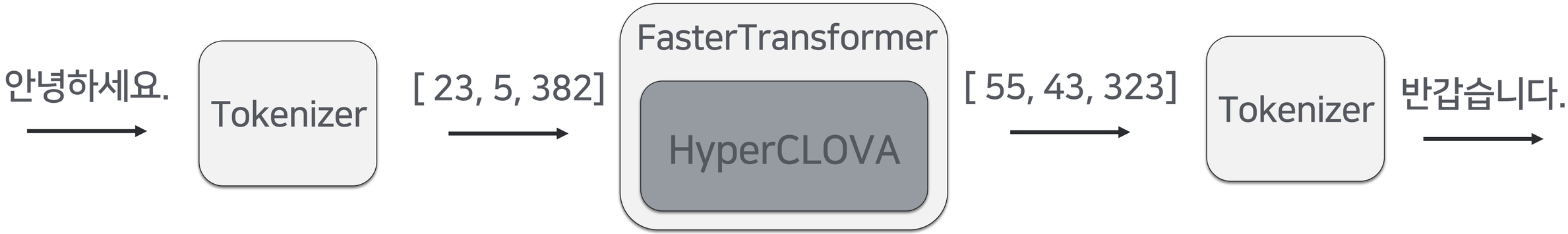
- 모델 로딩
- 데이터 전달
- 프로세스 관리

- 인퍼런스 (FT)
- 후처리

3.5 FTserve: Completion

Completion : 전처리, 인퍼런스, 후처리 담당

- 기본적으로 HyperCLOVA는 문장을 입력하면 뒤에 이어질 문장을 출력해준다.
- 하지만 실제로 입 출력 되는 것은 토큰 ID(숫자)들이다.
- Tokenizer를 이용해서 처리해 주어야한다.



토크나이저 외에도…

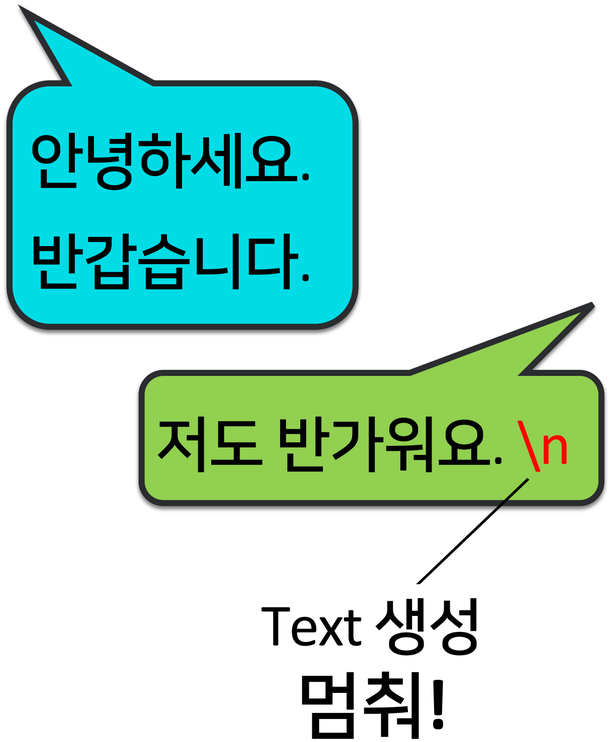
- 문자열 예외 처리(빈 문자열, 너무 긴 문자열..)
- Custom vector 입력(prompt tuning), attention mask 준비 등 다양한 기능을 담당

4. 서비스 기능 구현

개발 목표

- 서비스에 필요한 추가 기능들 구현

- Early Stop



- Semantic Search

- Prompt Tuning

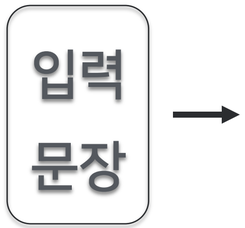
최신 연구 내용 적용

The power of scale for parameter-efficient prompt tuning
B Lester, R Al-Rfou, N Constant - arXiv preprint arXiv:2104.08691, 2021 - arxiv.org
In this work, we explore" **prompt tuning**", a simple yet effective mechanism for learning" soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation …
☆ 99 31회 인용 관련 학술자료 전체 2개의 버전 »

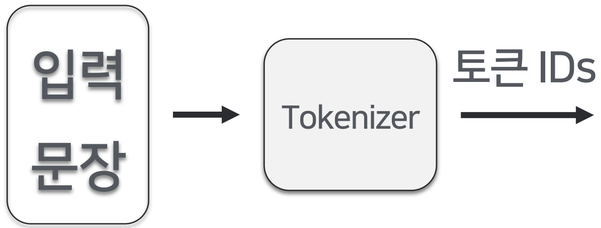
Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification
S Hu, N Ding, H Wang, Z Liu, J Li, M Sun - arXiv preprint arXiv:2108.02035, 2021 - arxiv.org
Tuning pre-trained language models (PLMs) with task-specific prompts has been a promising approach for text classification. Particularly, previous studies suggest that **prompt-tuning** has remarkable superiority in the low-data scenario over the generic fine-tuning …
☆ 99 6회 인용 전체 3개의 버전 »

PTR: Prompt Tuning with Rules for Text Classification
X Han, W Zhao, N Ding, Z Liu, M Sun - arXiv preprint arXiv:2105.11259, 2021 - arxiv.org
Fine-tuned pre-trained language models (PLMs) have achieved awesome performance on almost all NLP tasks. By using additional prompts to fine-tune PLMs, we can further stimulate the rich knowledge distributed in PLMs to better serve downstream task. **Prompt tuning** has …
☆ 99 13회 인용 관련 학술자료 전체 2개의 버전 »

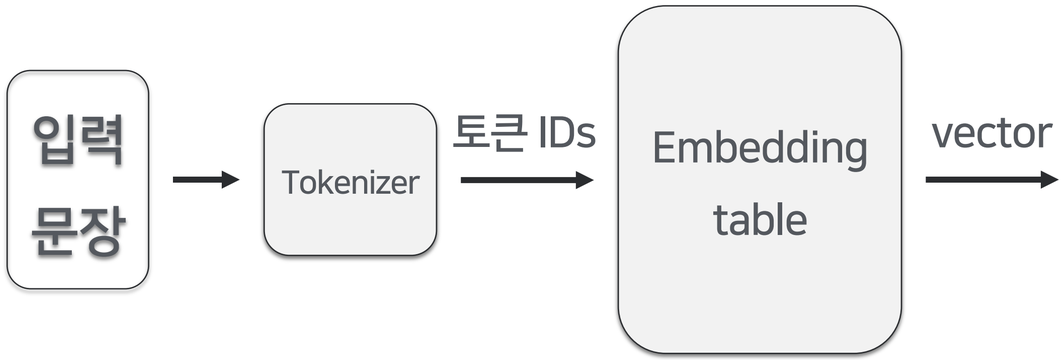
4.1 문장 생성 단계



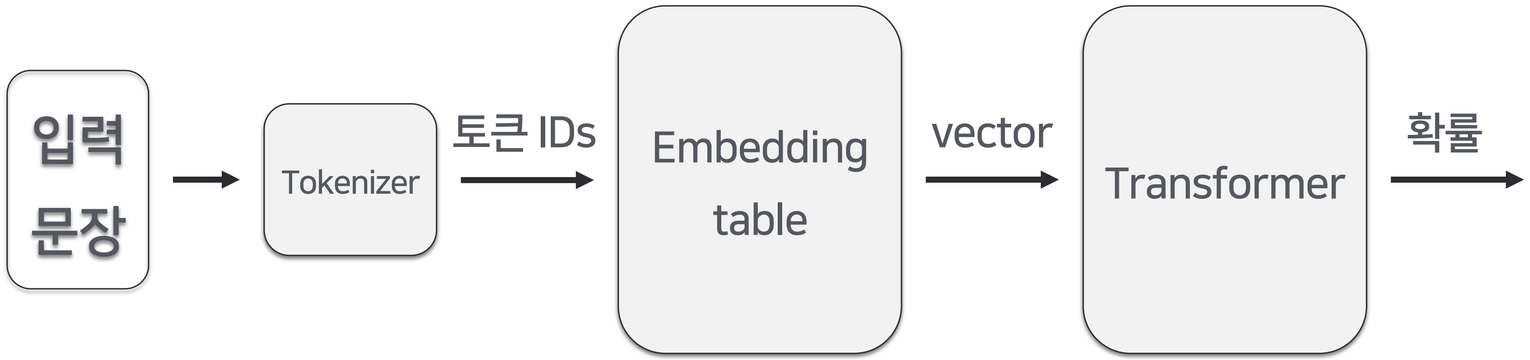
4.1 문장 생성 단계



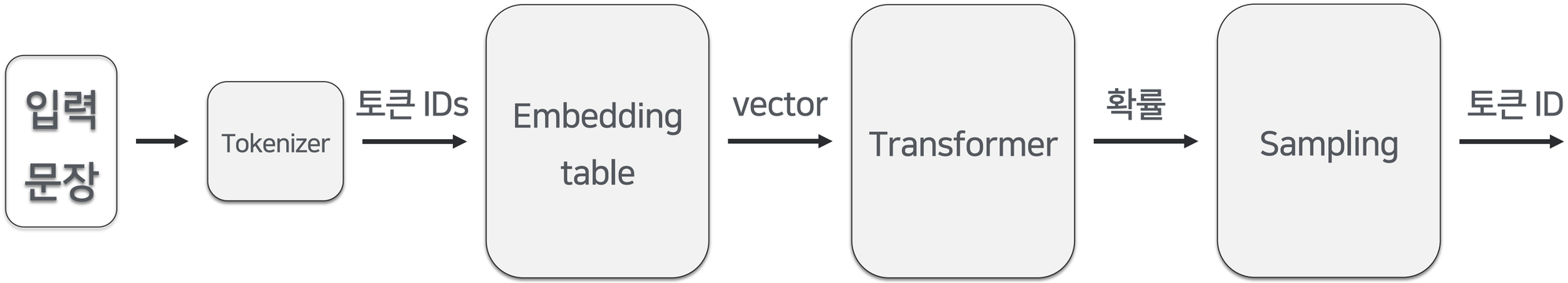
4.1 문장 생성 단계



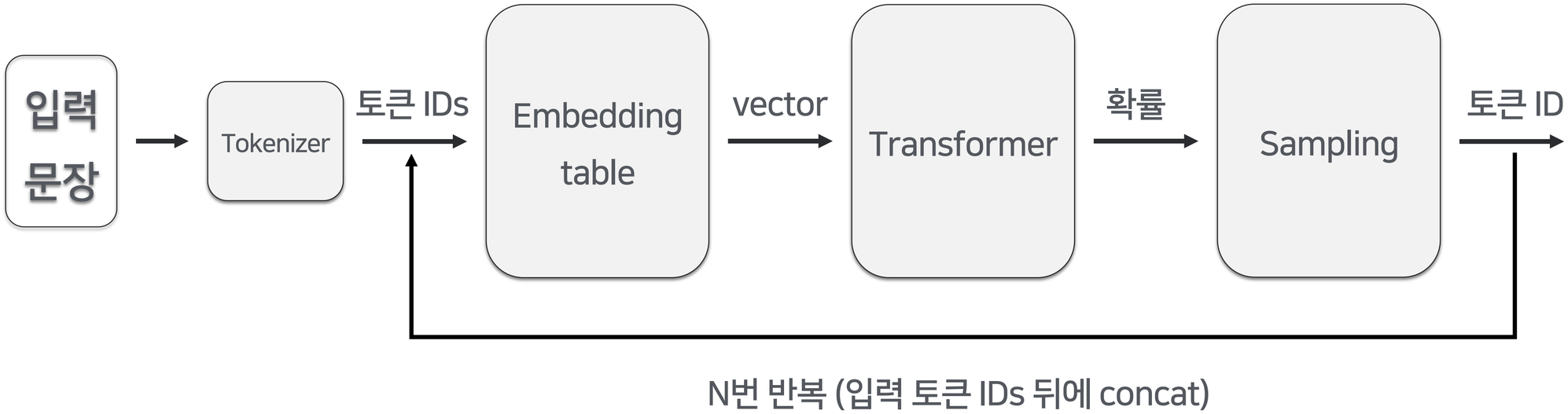
4.1 문장 생성 단계


입력 문장
Tokenizer
토큰 IDs
Embedding table
vector
Transformer
확률

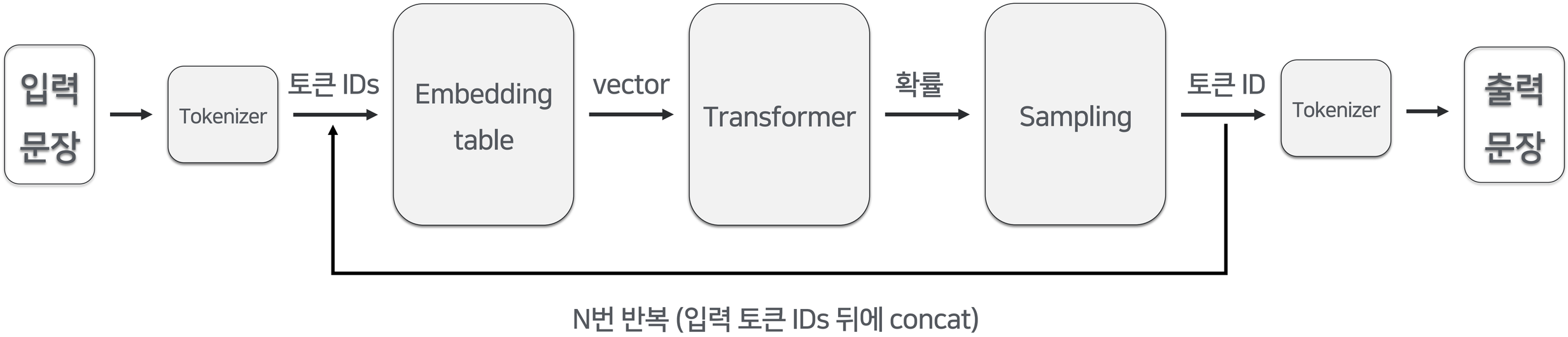
4.1 문장 생성 단계


입력 문장
Tokenizer
토큰 IDs
Embedding table
vector
Transformer
확률
Sampling
토큰 ID

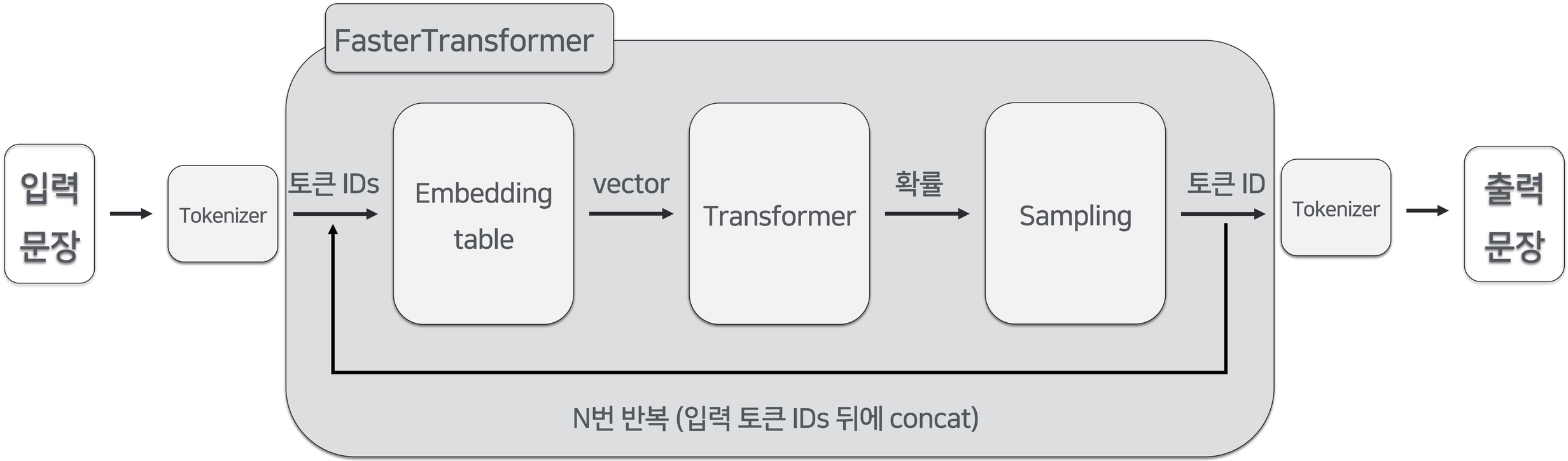
4.1 문장 생성 단계


입력 문장
Tokenizer
토큰 IDs
Embedding table
vector
Transformer
확률
Sampling
토큰 ID
N번 반복 (입력 토큰 IDs 뒤에 concat)

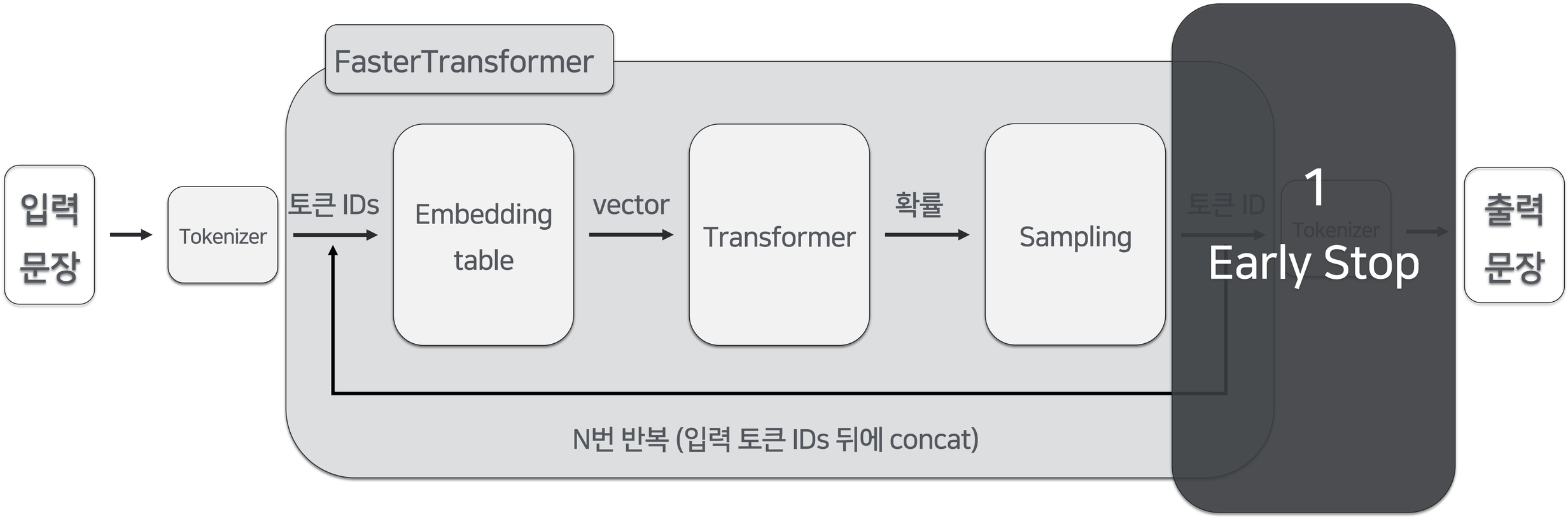
4.1 문장 생성 단계


입력 문장
Tokenizer
토큰 IDs
Embedding table
vector
Transformer
확률
Sampling
토큰 ID
Tokenizer
출력 문장
N번 반복 (입력 토큰 IDs 뒤에 concat)

4.1 문장 생성 단계


FasterTransformer
입력 문장
Tokenizer
토큰 IDs
Embedding table
vector
Transformer
확률
Sampling
토큰 ID
Tokenizer
출력 문장
N번 반복 (입력 토큰 IDs 뒤에 concat)

4.2 Early Stop



4.2 Early Stop

DEVIEW 2021

Early stop의 필요성

- FasterTransformer의 특성상 요청한 개수만큼 토큰 생성
- 요청 개수가 작을 때: 원하는 만큼 문장 생성이 안될 수 있음
- 요청 개수가 클 때 : 너무 오랜 시간이 소요됨

4.2 Early Stop

DEVIEW 2021

Early stop의 필요성

- FasterTransformer의 특성상 요청한 개수만큼 토큰 생성
- 요청 개수가 작을 때: 원하는 만큼 문장 생성이 안될 수 있음
- 요청 개수가 클 때 : 너무 오랜 시간이 소요됨

⇒딱 필요한 만큼만 생성할 순 없을까?

⇒패턴이 일치하면 생성 종료!

4.2 Early Stop

Example)

사용자: 안녕 클로바

클로바: 네 안녕하세요 반갑습니다.

사용자: 너는 취미가 뭐야?

32 토큰 요청

HyperCLOVA

4.2 Early Stop

Example)

사용자: 안녕 클로바
클로바: 네 안녕하세요 반갑습니다.
사용자: 너는 취미가 뭐야?

32 토큰 요청

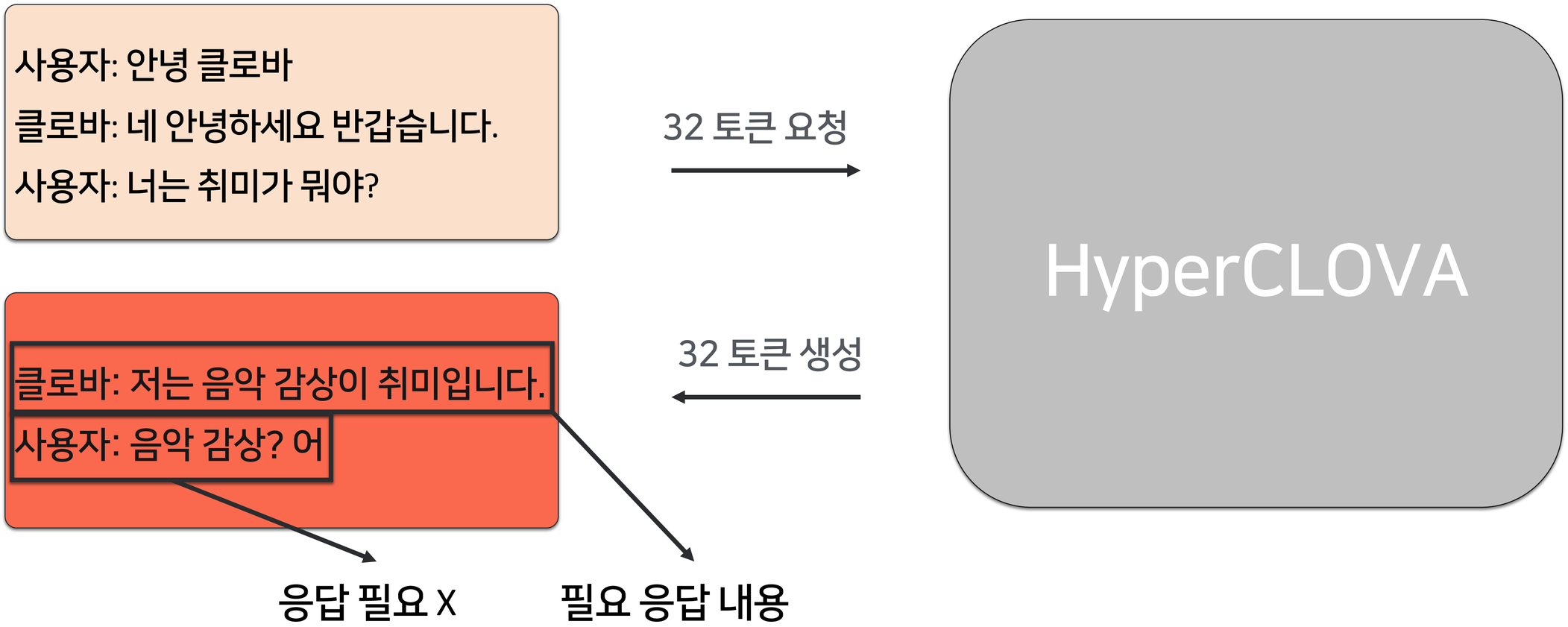
HyperCLOVA

32 토큰 생성

클로바: 저는 음악 감상이 취미입니다.
사용자: 음악 감상? 어

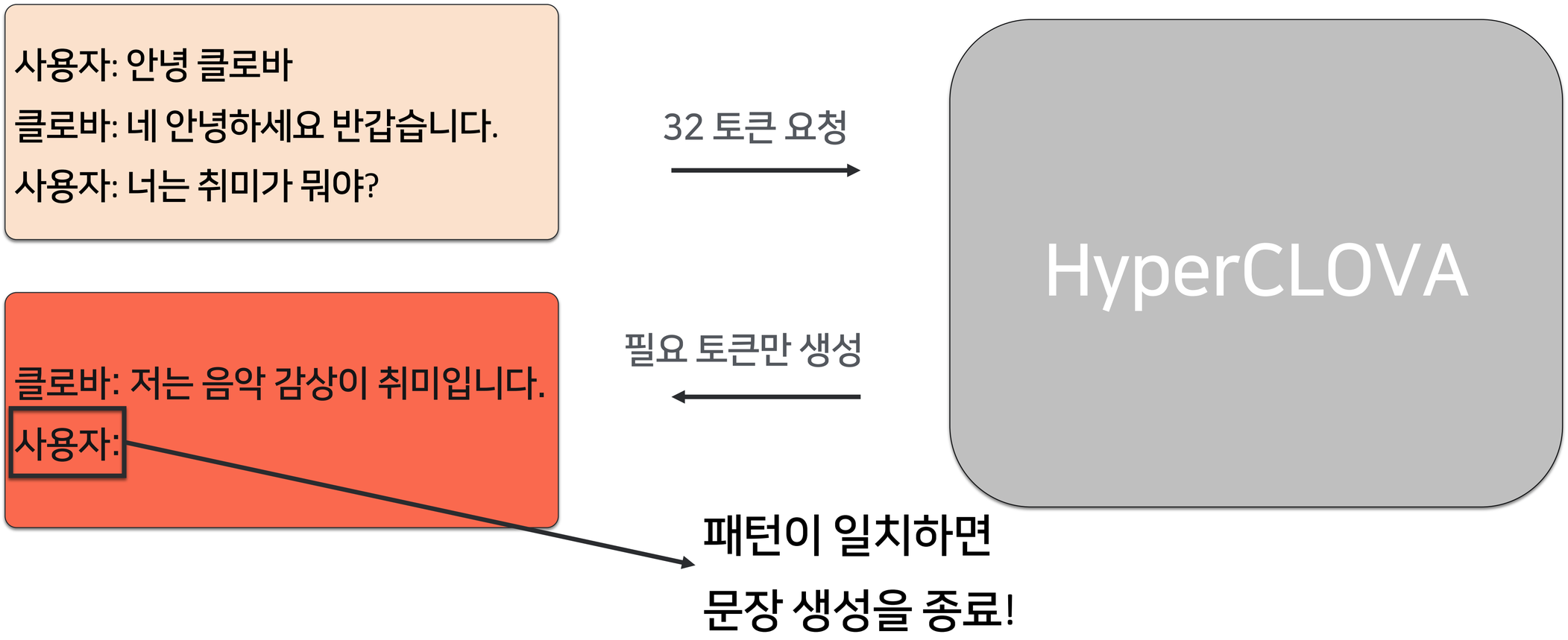
4.2 Early Stop

Example)

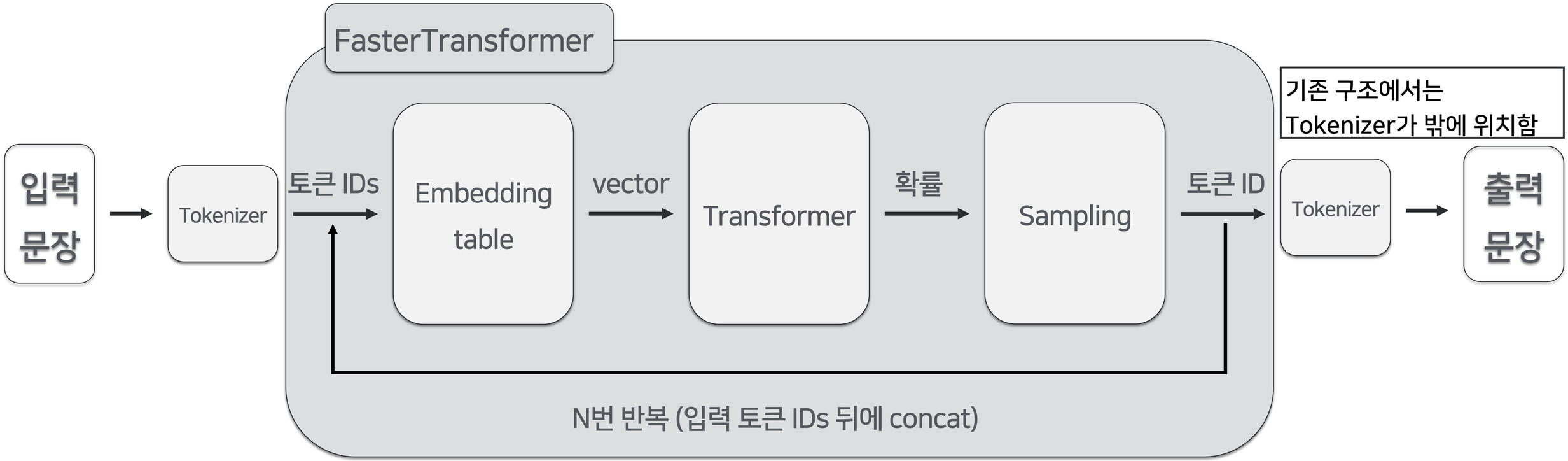


4.2 Early Stop

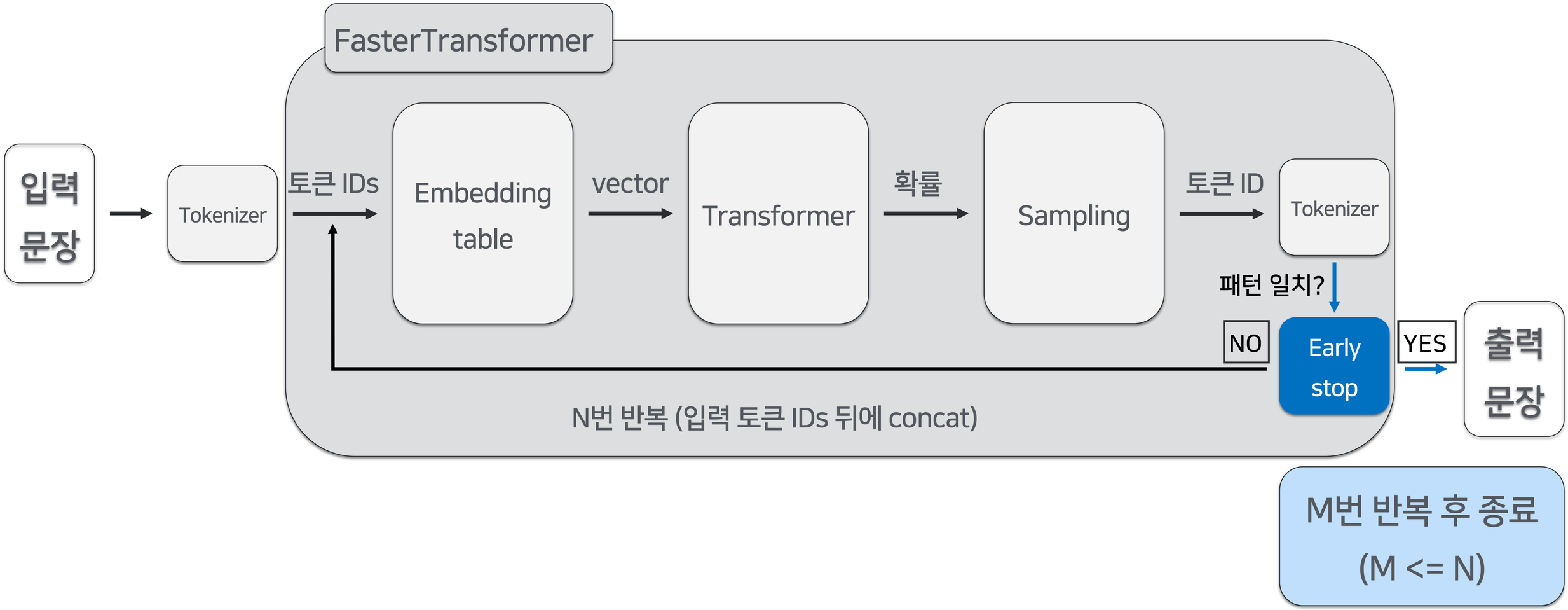
Example)



4.2 Early Stop



4.2 Early Stop



4.2 Early Stop

FasterTransformer 내부에서 토큰 ID를 문자열로 변환 후 패턴 검색

ID	vocab
1	ìłķì¹Í
2	ëĦĺìĸ
3	íķŃê³µ
….	
10000	íķĺëĵłëį°

Decoder (BPE tokenizer)

4.2 Early Stop

FasterTransformer 내부에서 토큰 ID를 문자열로 변환 후 패턴 검색

[3, 2, 10000] →

ID	vocab
1	ìłķì¹Í
2	ëĦĺìĸ
3	íķŃê³µ
….	
10000	íķĺëĵłëį°

Decoder (BPE tokenizer)

4.2 Early Stop

FasterTransformer 내부에서 토큰 ID를 문자열로 변환 후 패턴 검색

[3, 2, 10000] →

ID	vocab
1	ìłķì¹Í
2	ëĦĺìĸ
3	íķŃê³µ
….	
10000	íķĺëĵłëį°

→

3 2 10000

íķŃê³µ	ëĦĺìĸ	íķĺëĵłëį

Decoder (BPE tokenizer)

4.2 Early Stop

FasterTransformer 내부에서 토큰 ID를 문자열로 변환 후 패턴 검색

[3, 2, 10000] →

ID	vocab
1	ìłķì¹Í
2	ëĦĺìĸ
3	íķŃê³µ
….	
10000	íķĺëĵłëį°

Decoder (BPE tokenizer)

→

3 2 10000

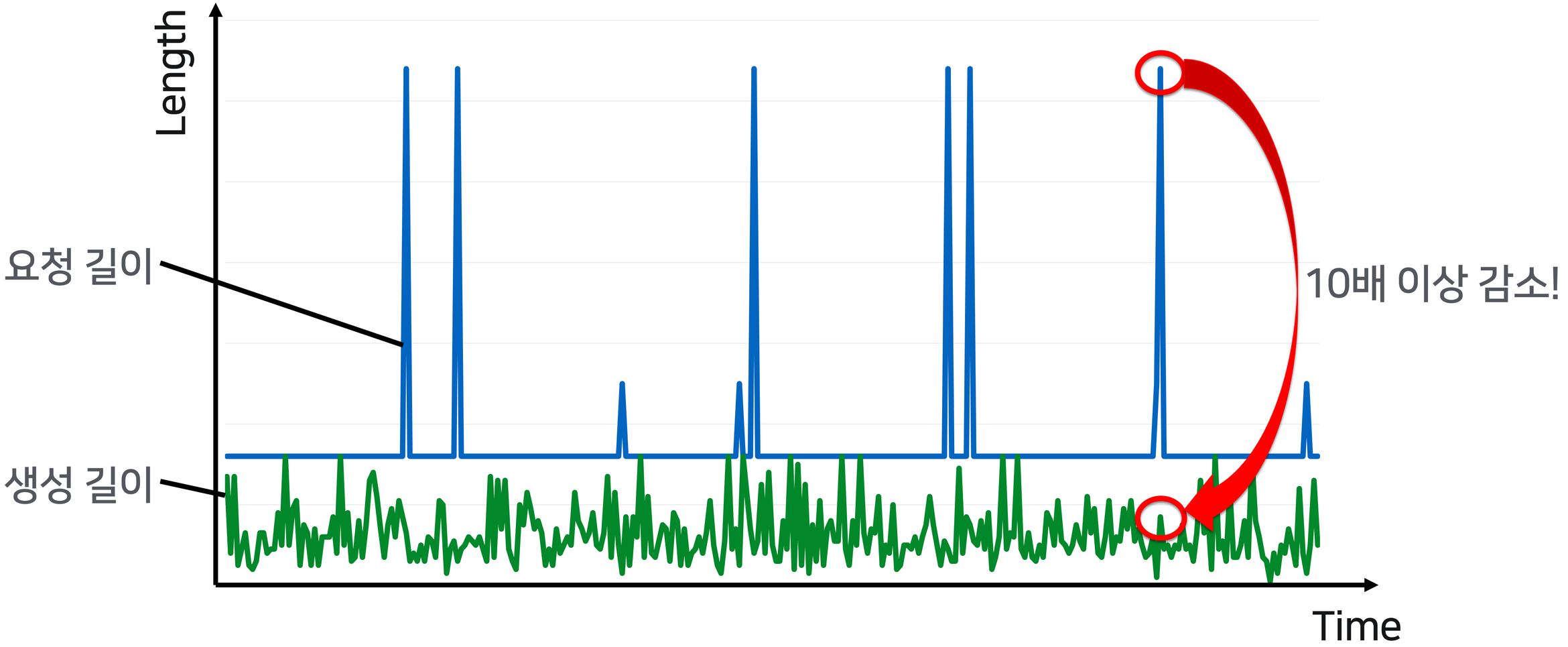
íķŃê³µ	ëĦĺìĸ	íķĺëĵłëį

→ 사용자

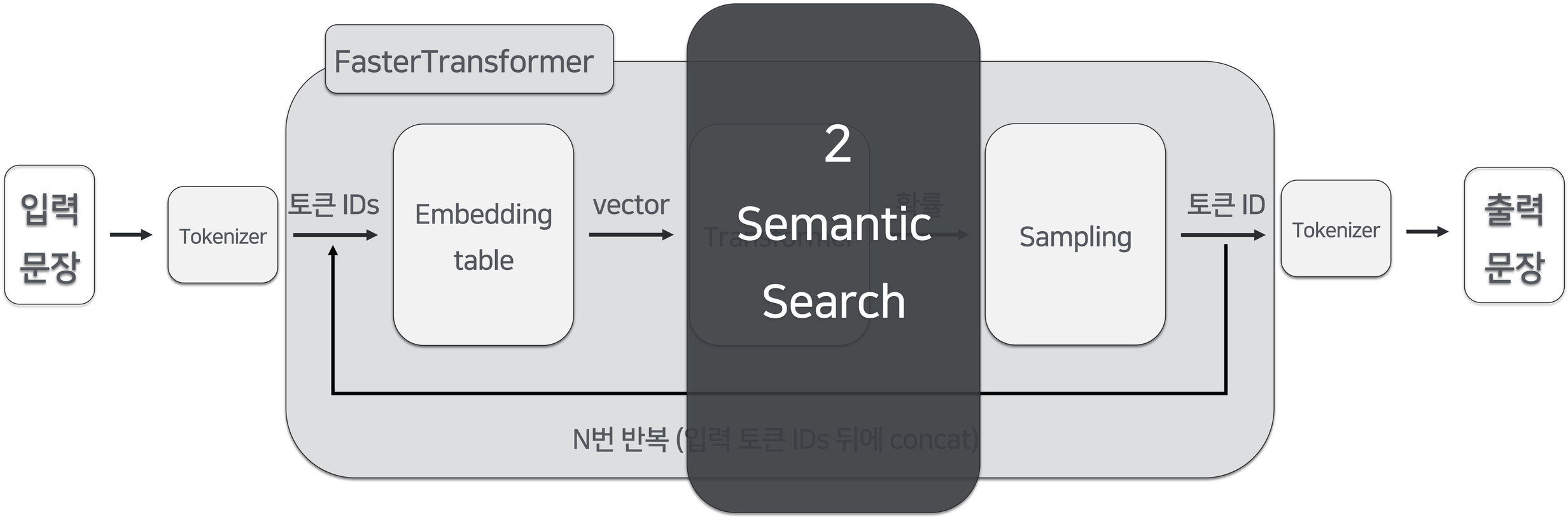
패턴 일치하여 문장생성 종료!

4.2 Early Stop

요청 길이 vs 생성 길이 (일부 서비스 로그)



4.3 Semantic Search



4.3 Semantic Search

DEVIEW 2021

Semantic Search

- 문서의 의미(시맨틱)을 분석해 검색하는 것을 말한다. 즉, 문장이나 단락에 기술된 주제를 파악하고 이를 대상으로 검색하는 것이라 할 수 있다. – 지식백과

- 주어진 문장(query)이 있을 때 여러 후보군 들 중(documents) 가장 적합한 것을 찾아내는 형태로 사용됨.

4.3 Semantic Search

DEVIEW 2021

Example)

Query

나는 어렸을 때부터 한식을 좋아했어.
지금도 내가 가장 좋아하는 음식은

Documents

- 된장찌개야
- 스파게티야
- 스테이크야
- 짜장면이야

Semantic search →

HyperCLOVA

4.3 Semantic Search

DEVIEW 2021

Example)

Query

나는 어렸을 때부터 한식을 좋아했어.
지금도 내가 가장 좋아하는 음식은

Documents

된장찌개야	:3.82
스파게티야	:5.92
스테이크야	:6.42
짜장면이야	:6.20

Semantic search →

← 각 Document의 스코어 계산 (낮을수록 좋음)

HyperCLOVA

4.3 Semantic Search

DEVIEW 2021

Example)

Query

나는 어렸을 때부터 한식을 좋아했어.
지금도 내가 가장 좋아하는 음식은

Documents

된장찌개야 :3.82

스파게티야 :5.92

스테이크야 :6.42

짜장면이야 :6.20

Semantic search →

← 각 Document의 스코어 계산 (낮을수록 좋음)

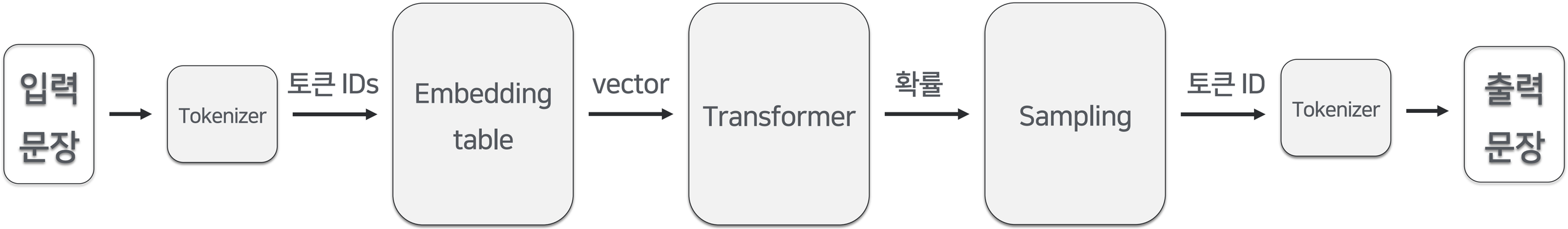
생성 모델이 어떻게???

HyperCLOVA

4.3 Semantic Search

DEVIEW 2021

문장 생성의 원리

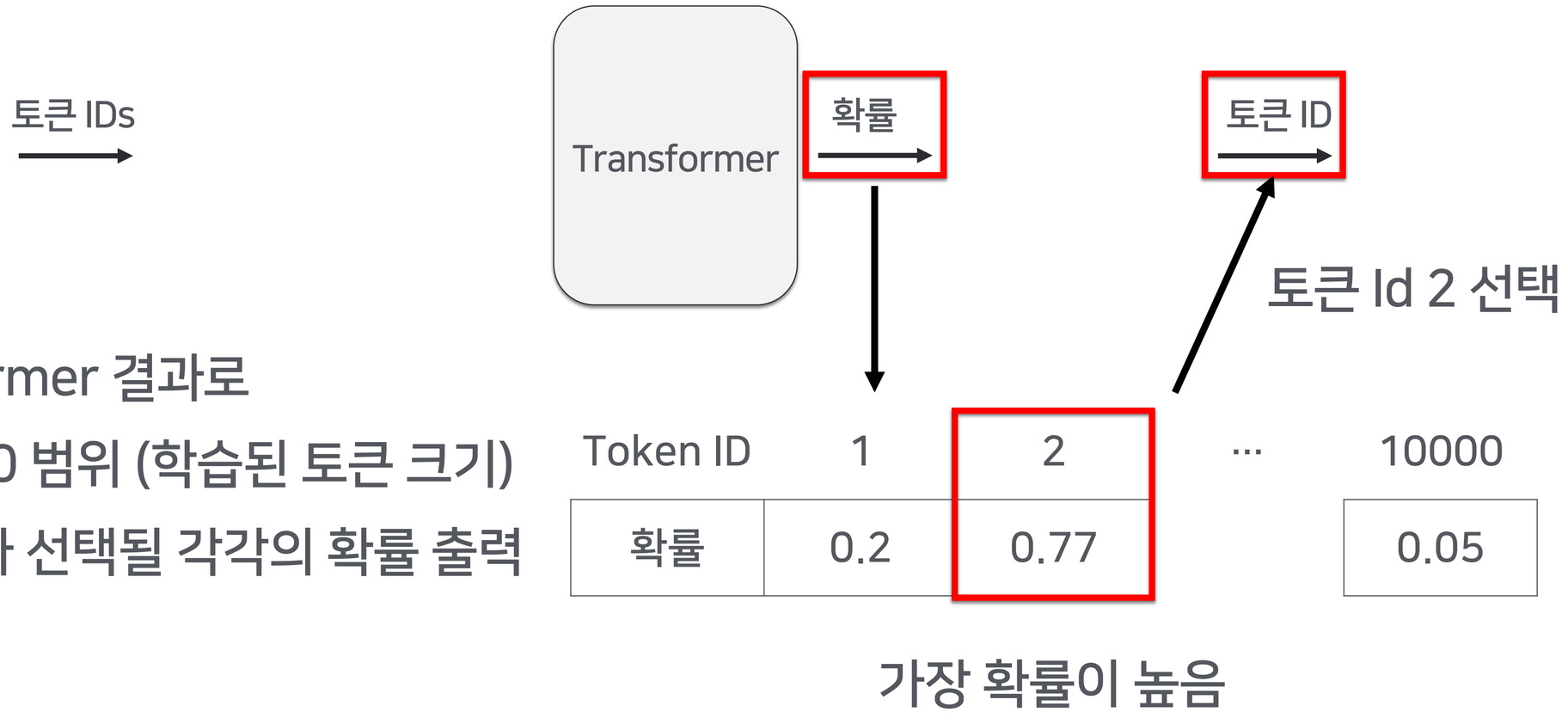


4.3 Semantic Search

문장 생성의 원리

4.3 Semantic Search

문장 생성의 원리



4.3 Semantic Search

Semantic Search?

Token ID	1	2	···	10000
확률	0.2	0.77		0.05

4.3 Semantic Search

Semantic Search?

Token ID	1	2	…	10000
확률	0.2	0.77		0.05

선택되지 않은 토큰 ID의 확률도 같이 계산된다!

4.3 Semantic Search

DEVIEW 2021

Semantic Search?

스파게티의 토큰 ID가
[1] 라면 1번 토큰 Id인
나올 확률이 0.2

	스파게티	된장	...	10000
Token ID	1	2		
확률	0.2	0.77		0.05

선택되지 않은 토큰 ID의 확률도 같이 계산된다!

4.3 Semantic Search

토큰 ID가 N개일 때는? (N>=2)

- 첫 번째 접근(generation stage) : 문장 생성 때처럼 위 과정을 N 번 반복한 뒤 각 토큰 확률의 음의 기하평균

4.3 Semantic Search

토큰 ID가 N개일 때는? (N>=2)

- 첫 번째 접근(generation stage) : 문장 생성 때처럼 위 과정을 N 번 반복한 뒤 각 토큰 확률의 음의 기하평균

⇒ 하지만, 너무나 느린 성능… 개선 방법은??

⇒ transformer의 동작 구조를 살펴보자!

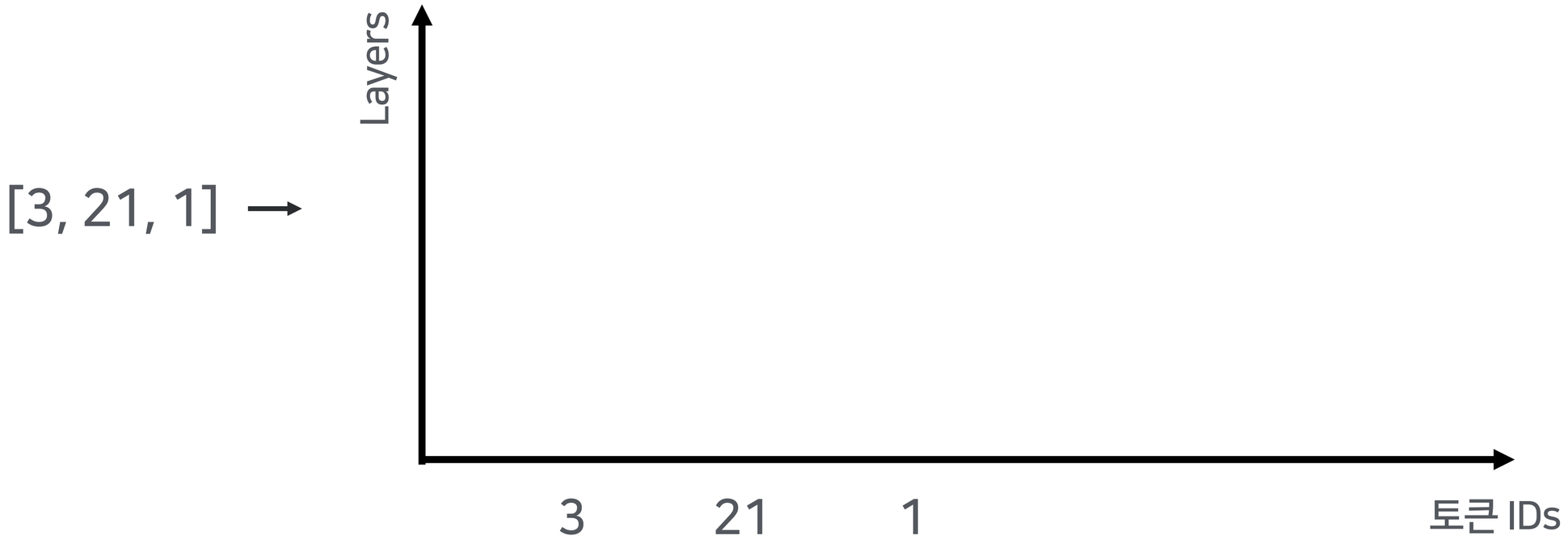
4.3 Semantic Search

DEVIEW 2021

Transformer의 동작 과정

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]



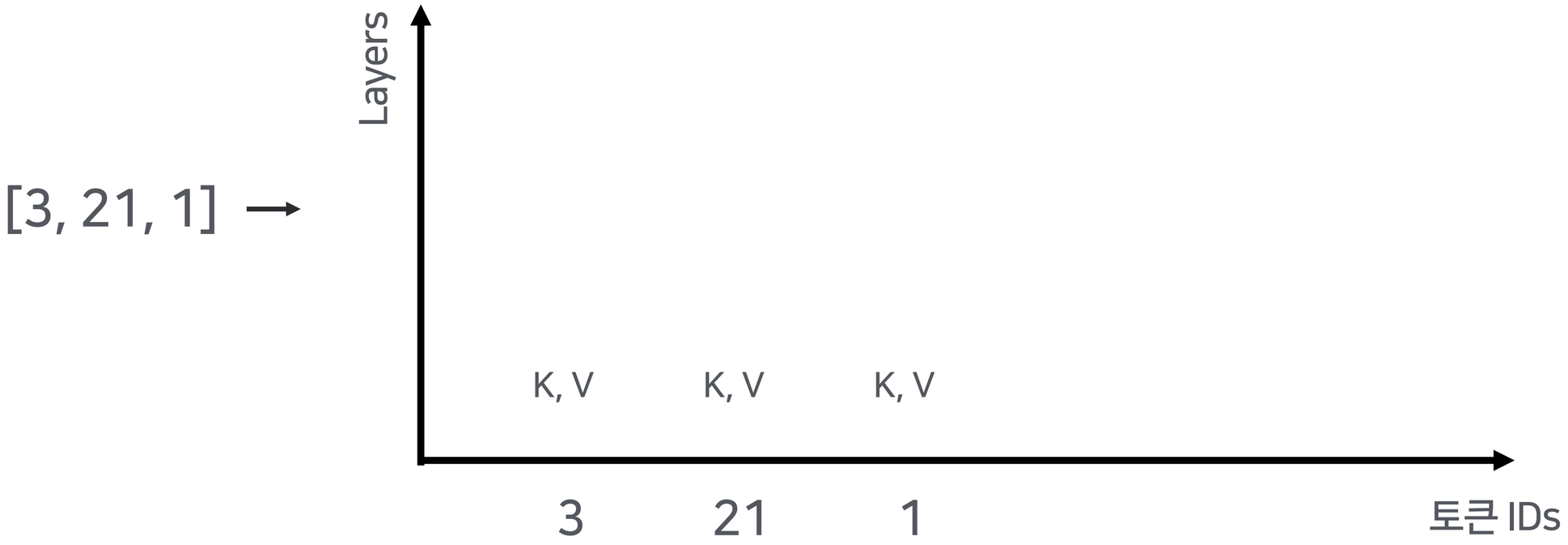
4.3 Semantic Search

DEVIEW 2021

Transformer의 동작 과정

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]



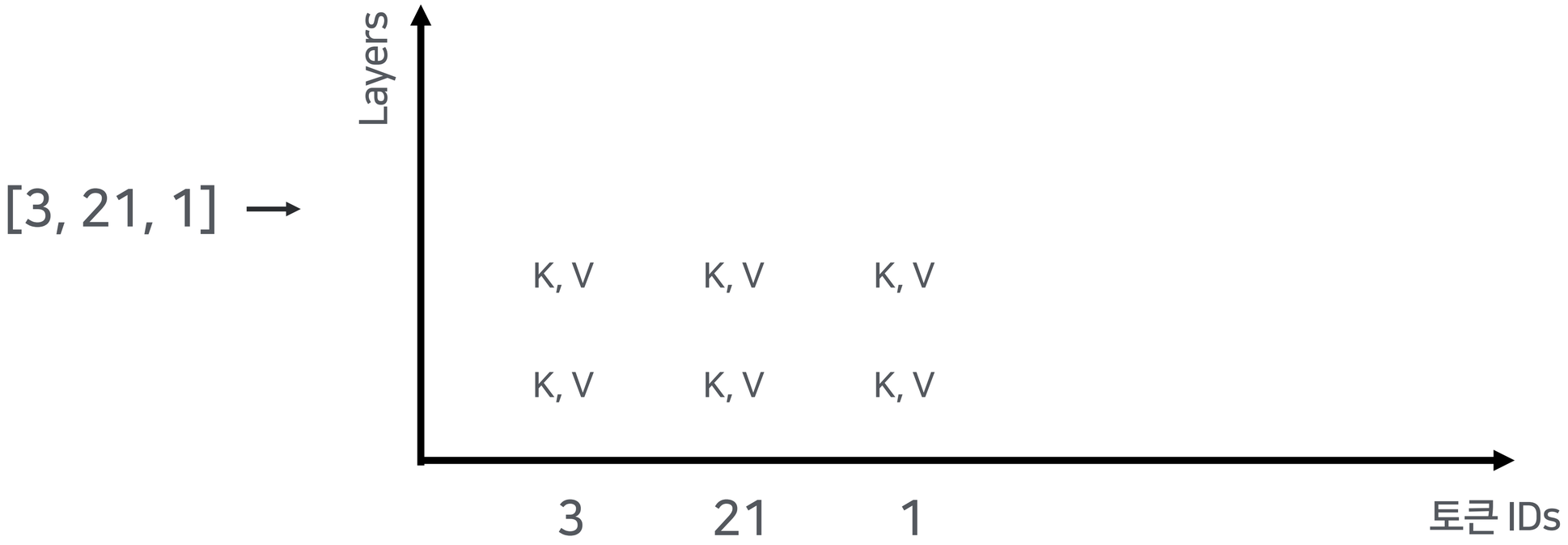
4.3 Semantic Search

DEVIEW 2021

Transformer의 동작 과정

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]



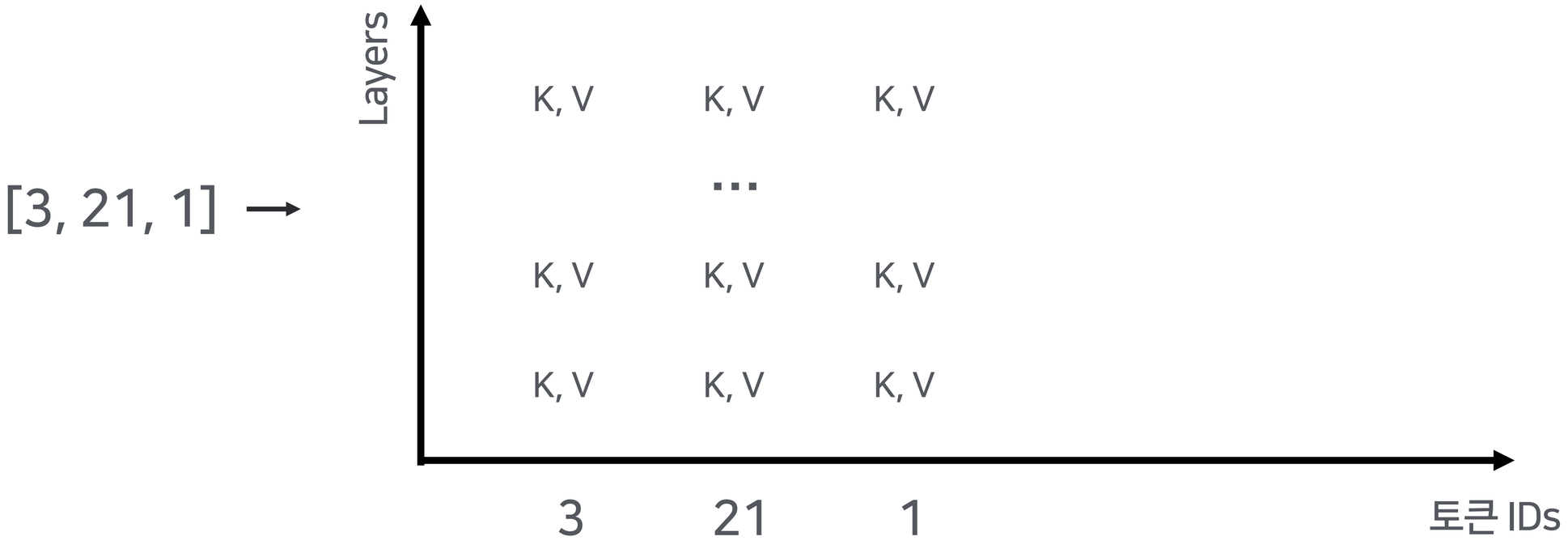
4.3 Semantic Search

DEVIEW 2021

Transformer의 동작 과정

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]

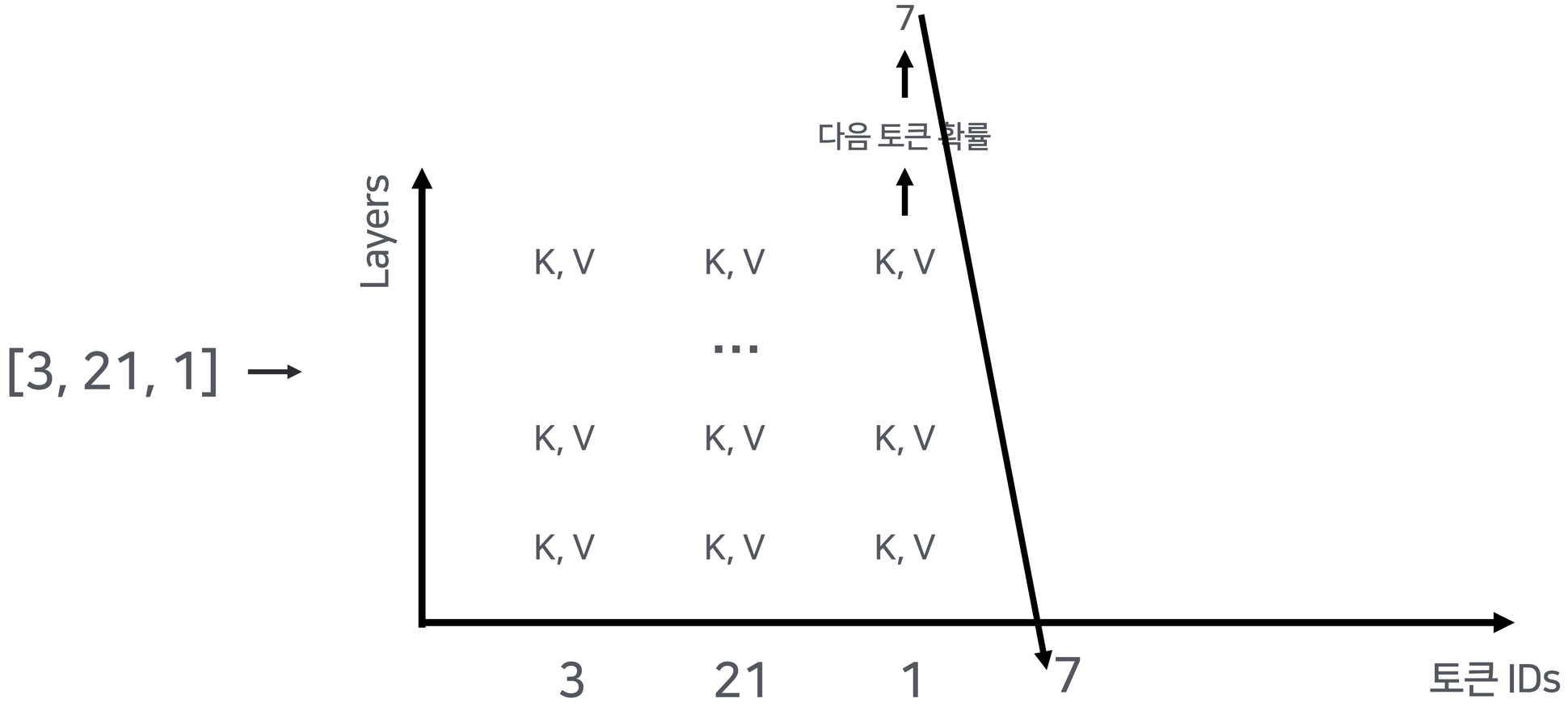


4.3 Semantic Search

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]

Transformer의 동작 과정



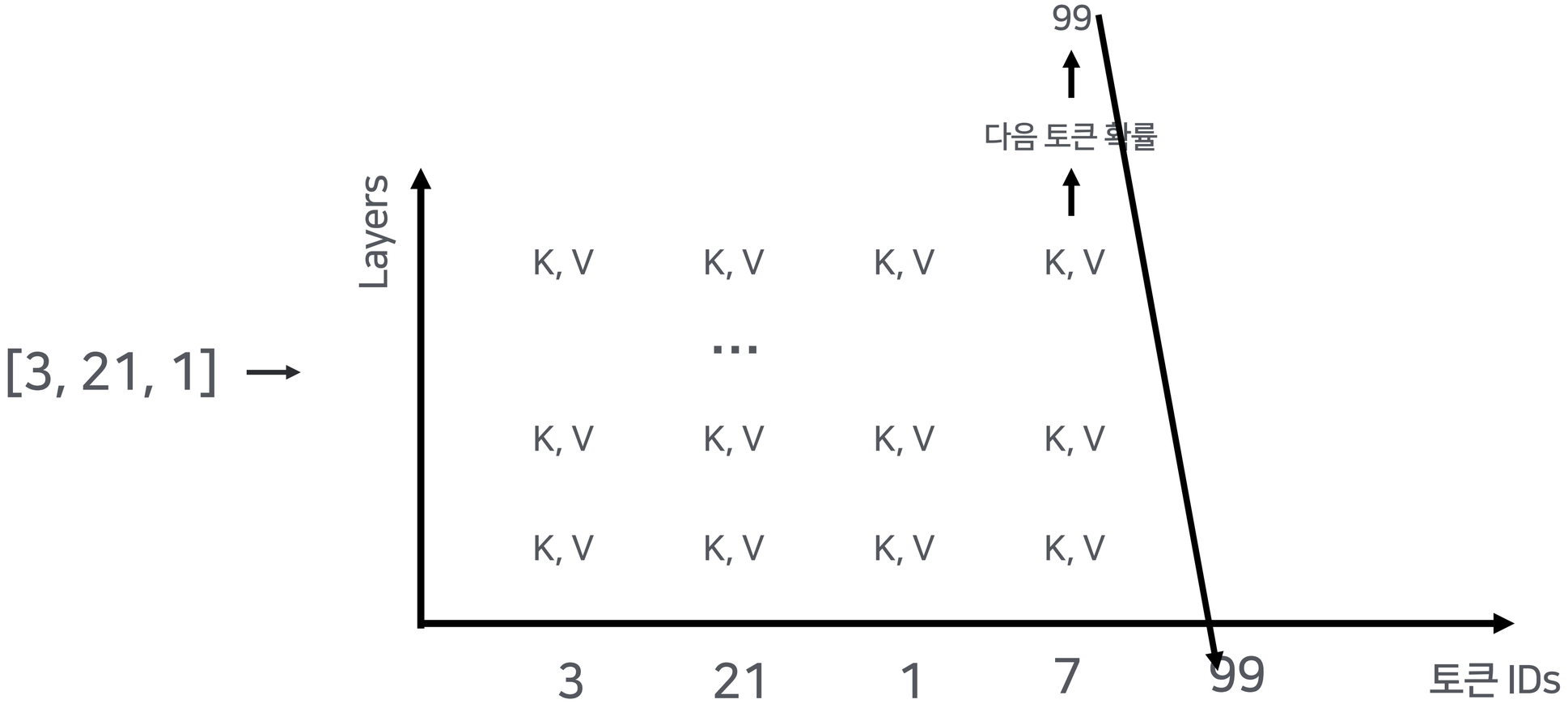
4.3 Semantic Search

DEVIEW 2021

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]

Transformer의 동작 과정



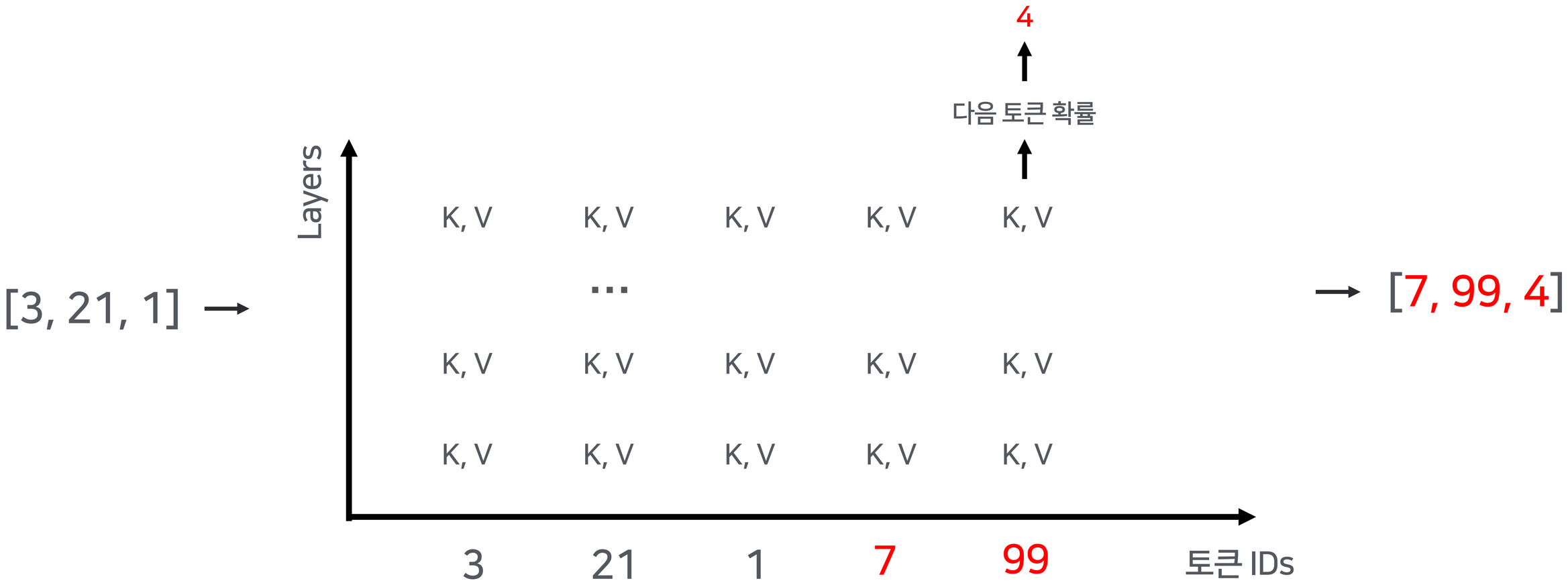
4.3 Semantic Search

DEVIEW 2021

Transformer의 동작 과정

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]



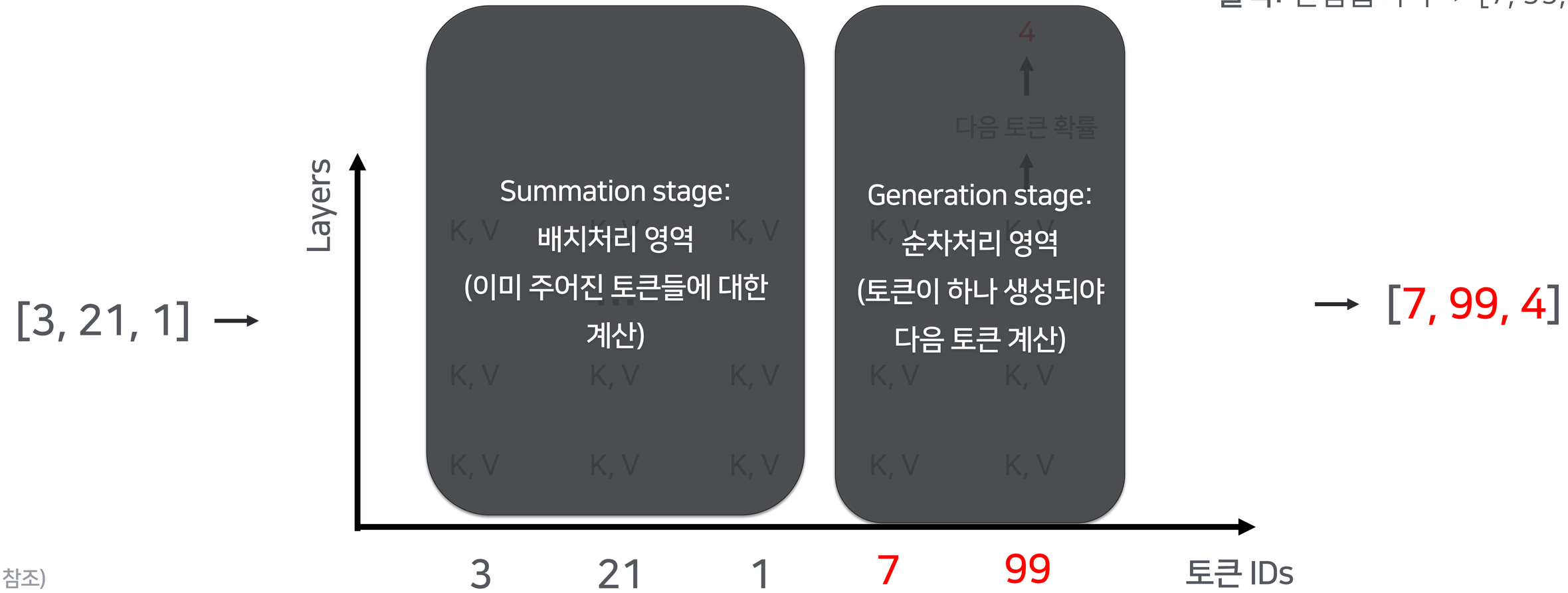
4.3 Semantic Search

DEVIEW 2021

Transformer의 동작 과정

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]



참조)

SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning (HPCA 2021) - https://arxiv.org/pdf/2012.09852.pdf

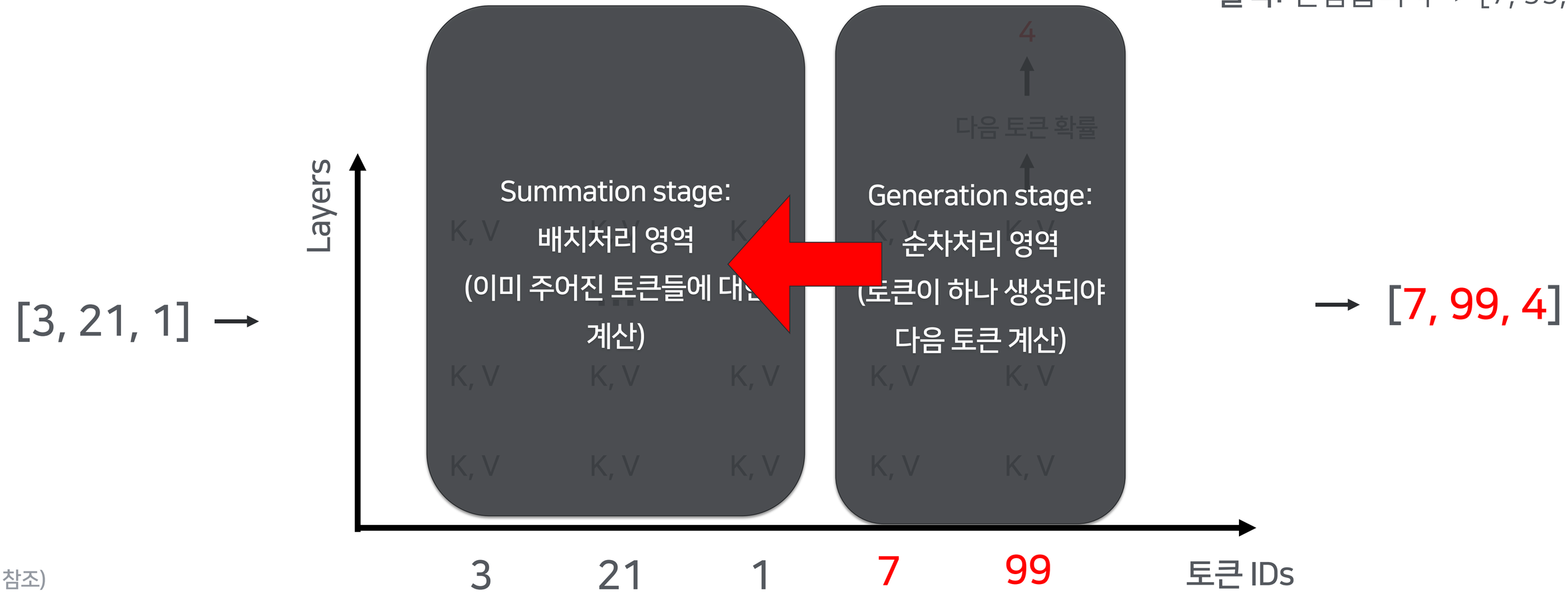
4.3 Semantic Search

DEVIEW 2021

Transformer의 동작 과정

입력: 안녕하세요 -> [3, 21, 1]

출력: 반갑습니다 -> [7, 99, 4]



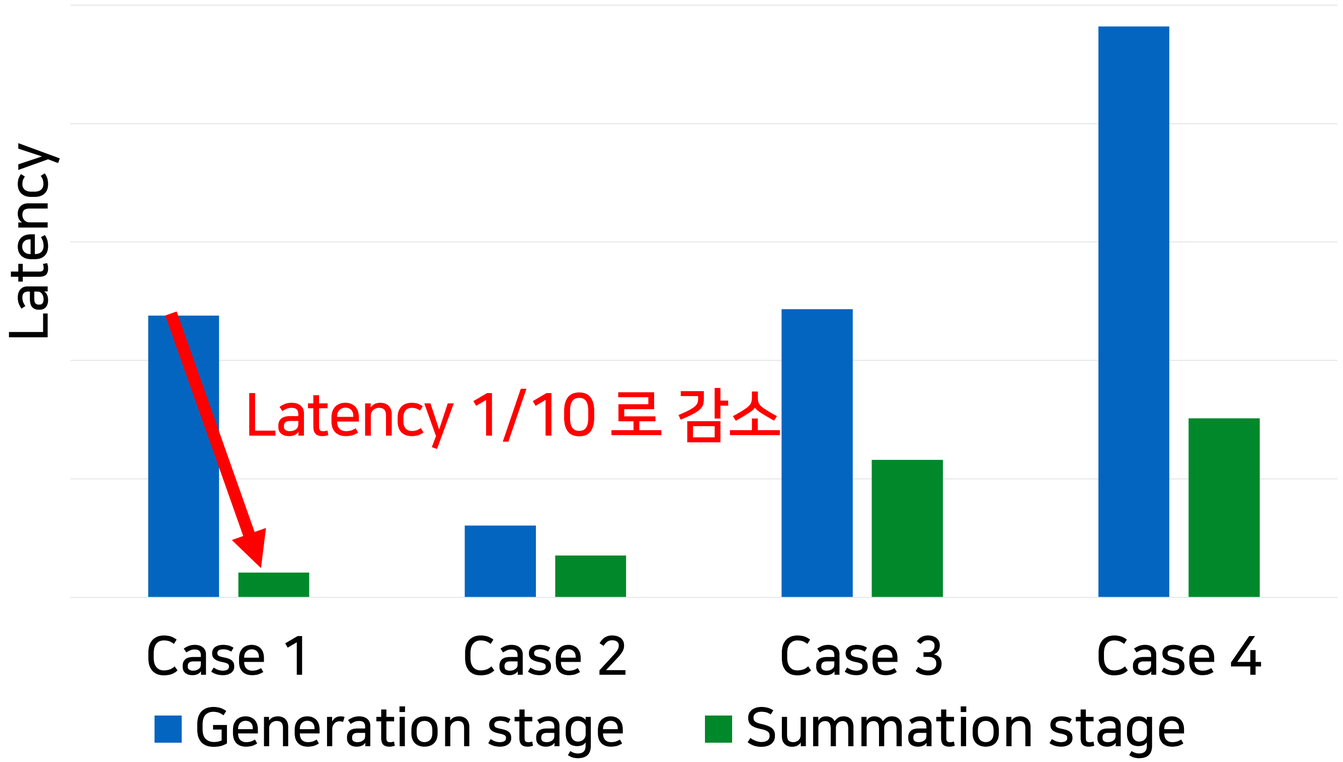
참조)

SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning (HPCA 2021) - https://arxiv.org/pdf/2012.09852.pdf

4.3 Semantic Search

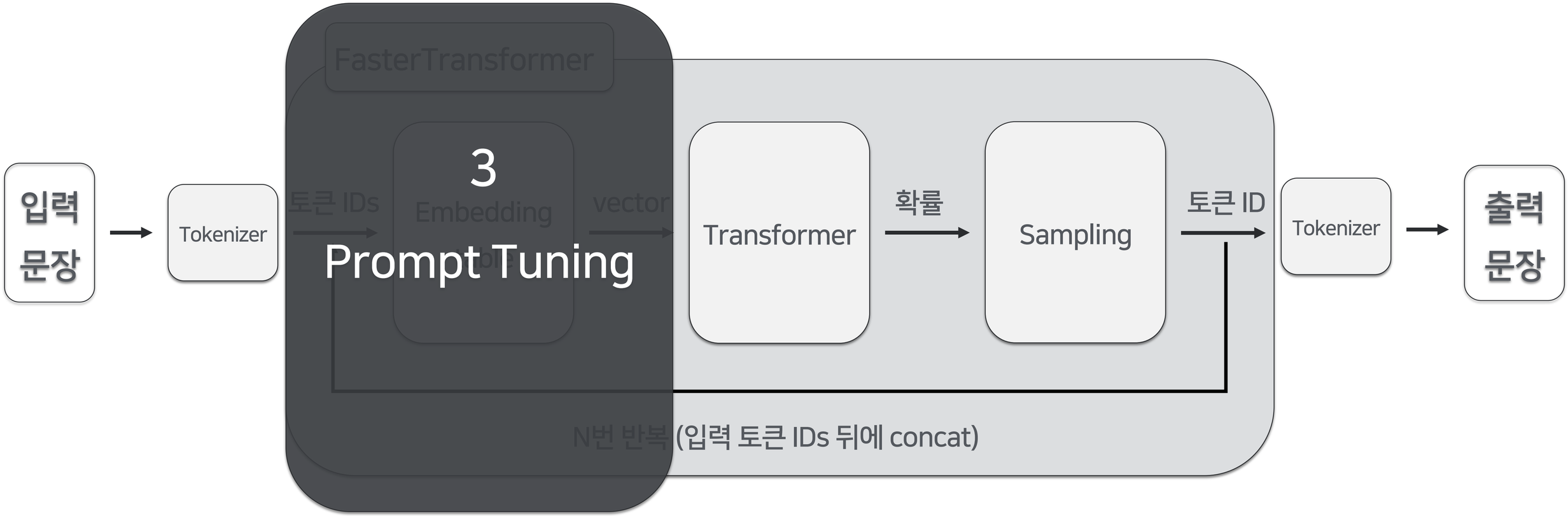
DEVIEW 2021

Summation Stage vs Generation Stage



	Query length	Document num	Document length
Case 1	2	2	36, 37
Case 2	7	4	7, 8, 5, 2
Case 3	508	2	8, 23
Case 4	431	3	48, 21, 49

4.4 Prompt Tuning

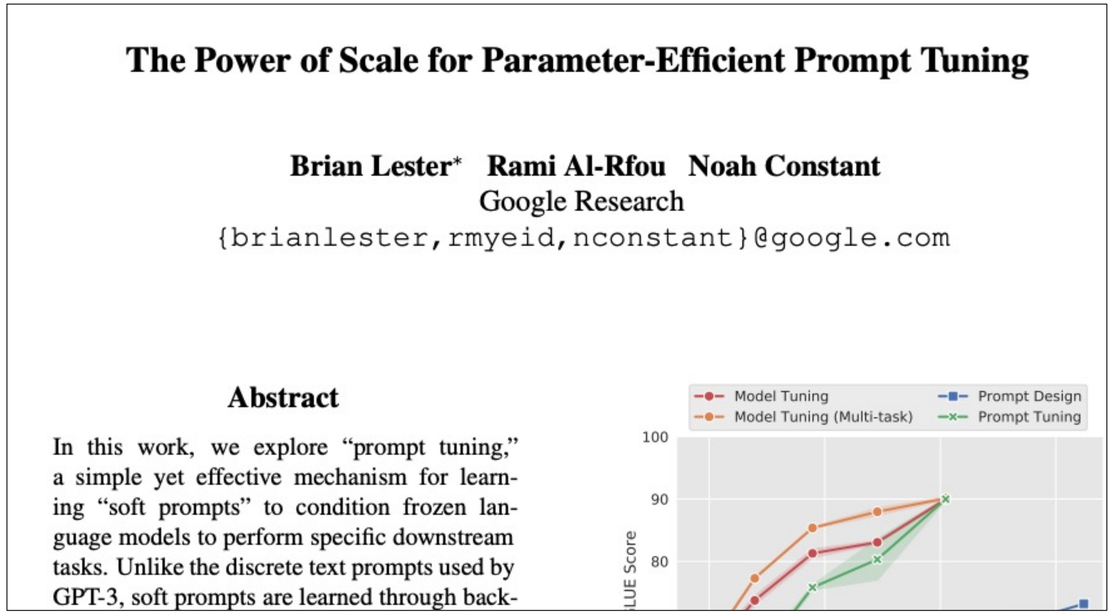


4.4 Prompt Tuning

DEVIEW 2021

서빙 관점에서 Prompt Tuning(P-Tuning) 이란?

- 특정 태스크에 맞게 학습된 vector를 inference 때 embedding table을 거치지 않고 **직접 주입**해주는 방법



The Power of Scale for Parameter-Efficient Prompt Tuning

Brian Lester* Rami Al-Rfou Noah Constant
Google Research
{brianlester,rmyeid,nconstant}@google.com

Abstract

In this work, we explore "prompt tuning," a simple yet effective mechanism for learning "soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through back-

참조)

The Power of Scale for Parameter-Efficient Prompt Tuning (EMNLP 2021) - https://arxiv.org/pdf/2104.08691.pdf

4.4 Prompt Tuning

P-Tuning의 효과

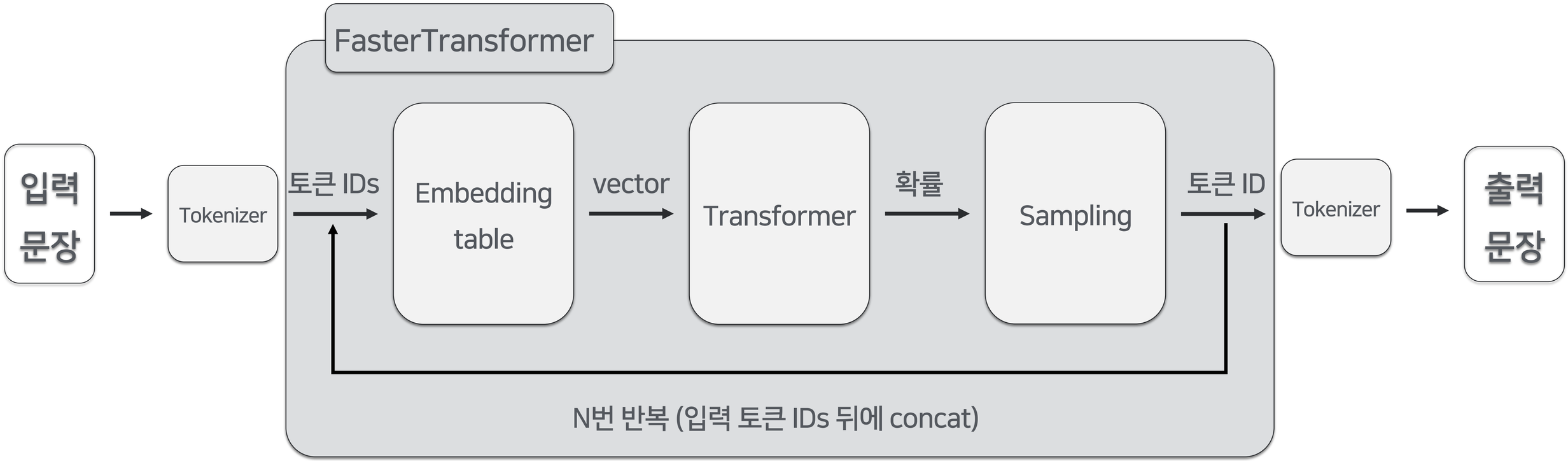
Q. 임진왜란은 언제 일어났어?

P-Tuning X : 조선시대에 일어났습니다

P-Tuning O : 1592년 조선시대에 일어났습니다

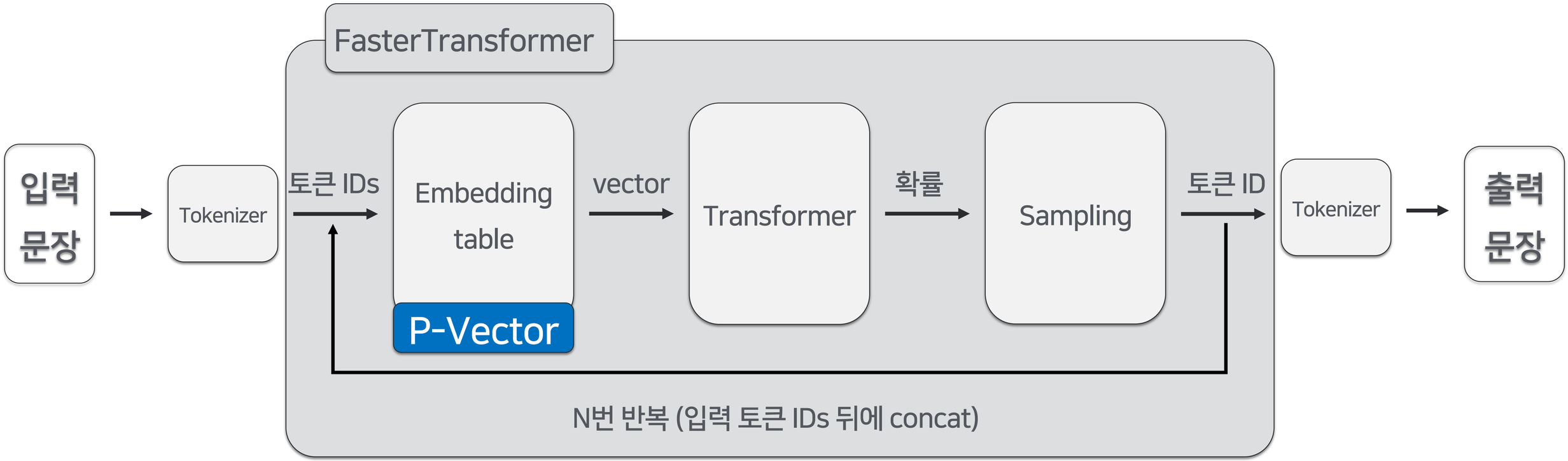
4.4 Prompt Tuning

첫번째 접근 방법



4.4 Prompt Tuning

첫번째 접근 방법: Embedding table에 P-Vector 등록



4.4 Prompt Tuning

첫번째 접근 방법: Embedding table에 P-Vector 등록

P-Vector 1: [0.11, 0.22, ···, 0.33]

P-Vector 2: [0.44, 0.55, ···, 0.66]

→

Embedding table

ID	vector
1	[0.01, 0.234 ··· 0.5]
2	[0.5, 0.0234 ··· 0.1]
3	[0.02, 0.51 ··· 0.73]
···.	[0.5, 0.234 ··· 0.5]
10000	[0.99, 0.14 ··· 0.8]

➡

Embedding table

ID	vector
1	[0.01, 0.234 ··· 0.5]
2	[0.5, 0.0234 ··· 0.1]
3	[0.02, 0.51 ··· 0.73]
···.	[0.5, 0.234 ··· 0.5]
10000	[0.99, 0.14 ··· 0.8]
10001	[0.11, 0.22 ···0.33]
10002	[0.44, 0.55 ··· 0.66]

4.4 Prompt Tuning

기대 효과

Embedding table

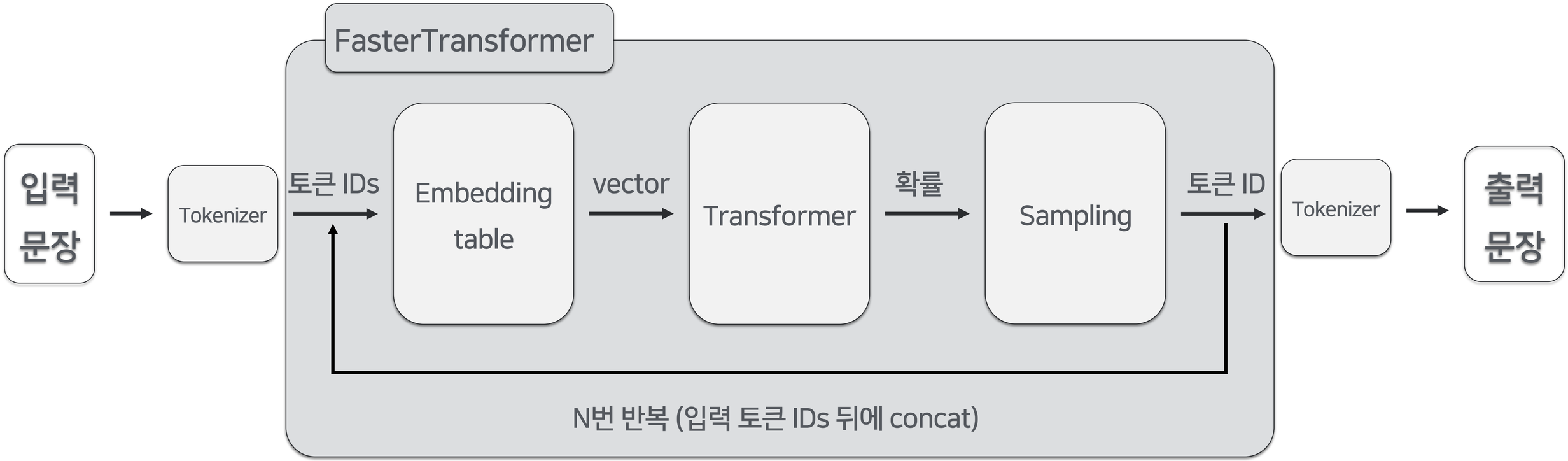
ID	vector
1	[0.01, 0.234 ··· 0.5]
2	[0.5, 0.0234 ··· 0.1]
3	[0.02, 0.51 ··· 0.73]
···.	[0.5, 0.234 ··· 0.5]
10000	[0.99, 0.14 ··· 0.8]
10001	[0.11, 0.22 ···0.33]
10002	[0.44, 0.55 ··· 0.66]

1. Caching
2. 커널 수정 X

생각치 못한 10% 정도의 latency 지연
(Embedding table 관련 gemm 연산)
+ 제한된 확장성

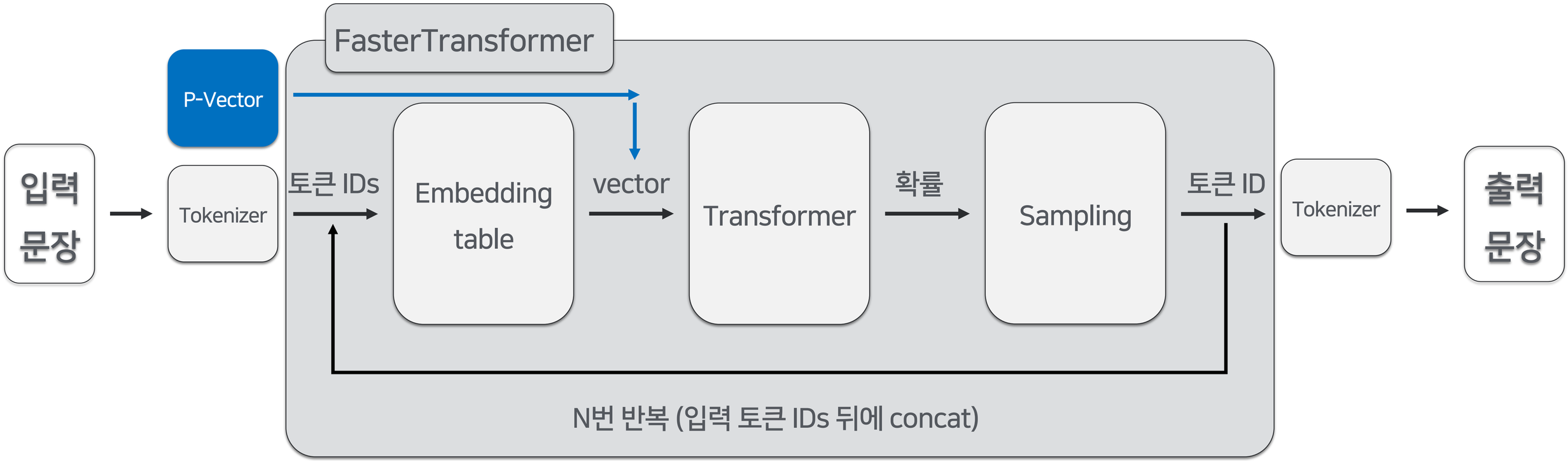
4.4 Prompt Tuning

두번째 접근 방법



4.4 Prompt Tuning

두번째 접근 방법 : P-Vector를 transformer로 직접 전달



4.4 Prompt Tuning

두번째 접근 방법: Lookup 커널 수정

Token ID : [3, 10001, 2, 10001]

P-Vector ID: [x, 1, x, 0]

Embedding table에 등록되지 않은 token ID는 prompt vector 취급

Embedding table

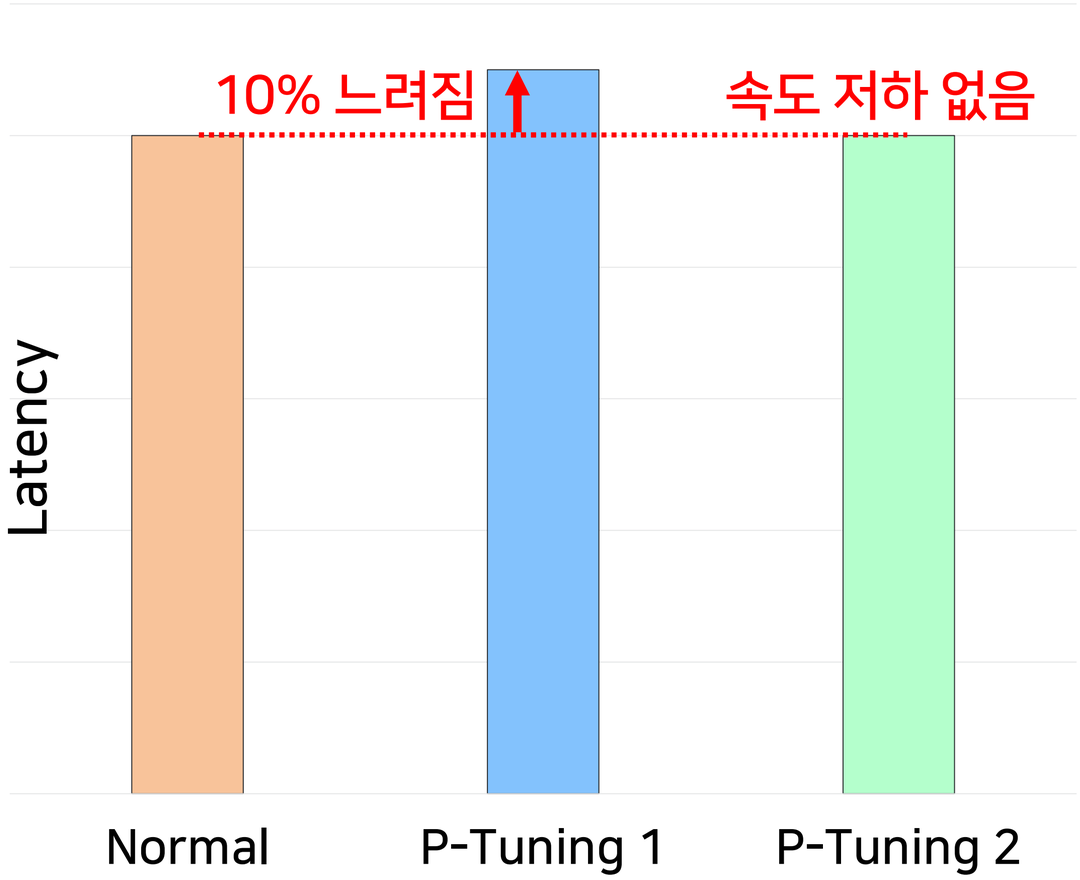
ID	vector
1	[0.01, 0.234 ··· 0.5]
2	[0.5, 0.0234 ··· 0.1]
3	[0.02, 0.51 ··· 0.73]
···.	[0.5, 0.234 ··· 0.5]
10000	[0.99, 0.14 ··· 0.8]

P-Vector

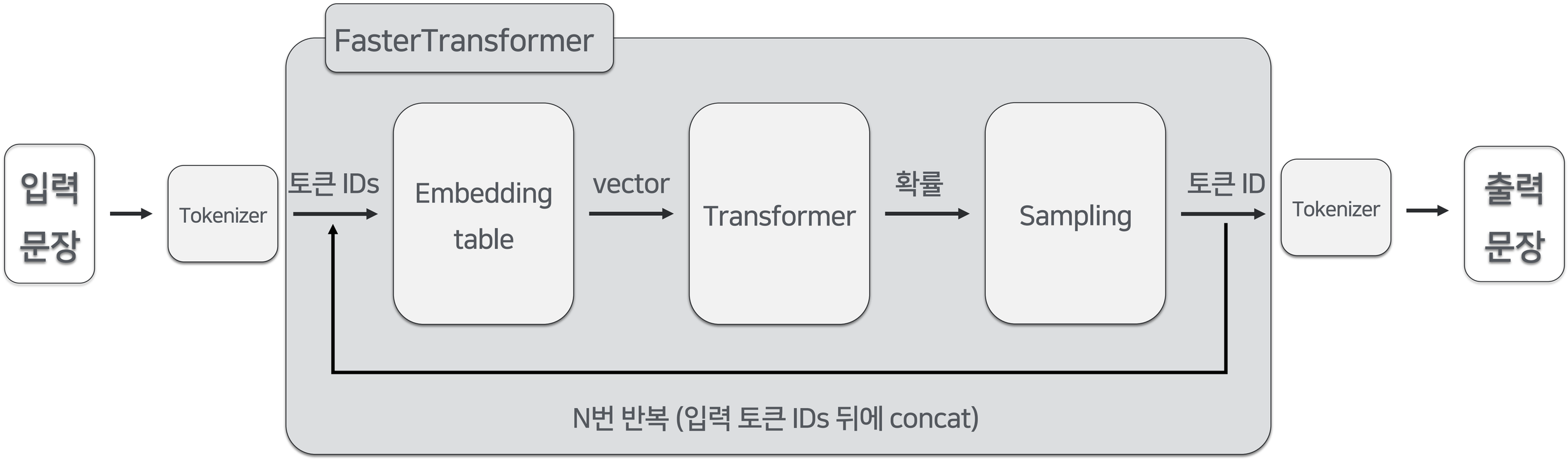
0	[0.11, 0.22 ···0.33]
1	[0.44, 0.55 ··· 0.66]

4.4 Prompt Tuning

응답 지연시간 평가



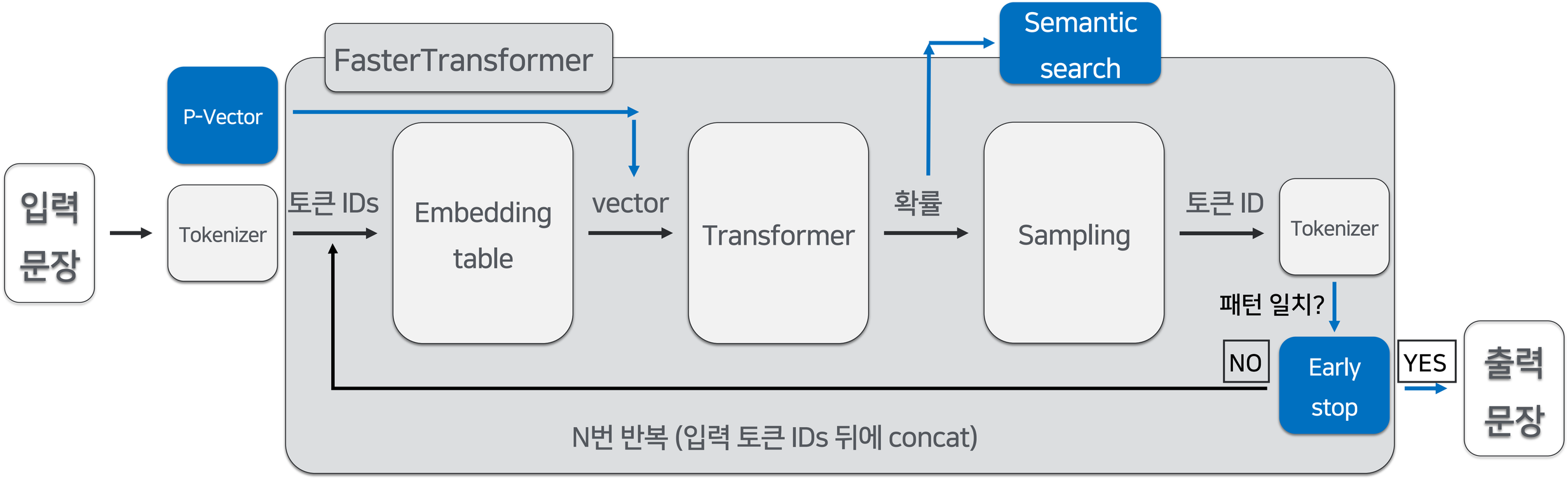
4.5 전체 구조


FasterTransformer
입력 문장
Tokenizer
토큰 IDs
Embedding table
vector
Transformer
확률
Sampling
토큰 ID
Tokenizer
출력 문장
N번 반복 (입력 토큰 IDs 뒤에 concat)

4.5 전체 구조

3. Prompt tuning 인퍼런스 지원

2. Semantic search 지원



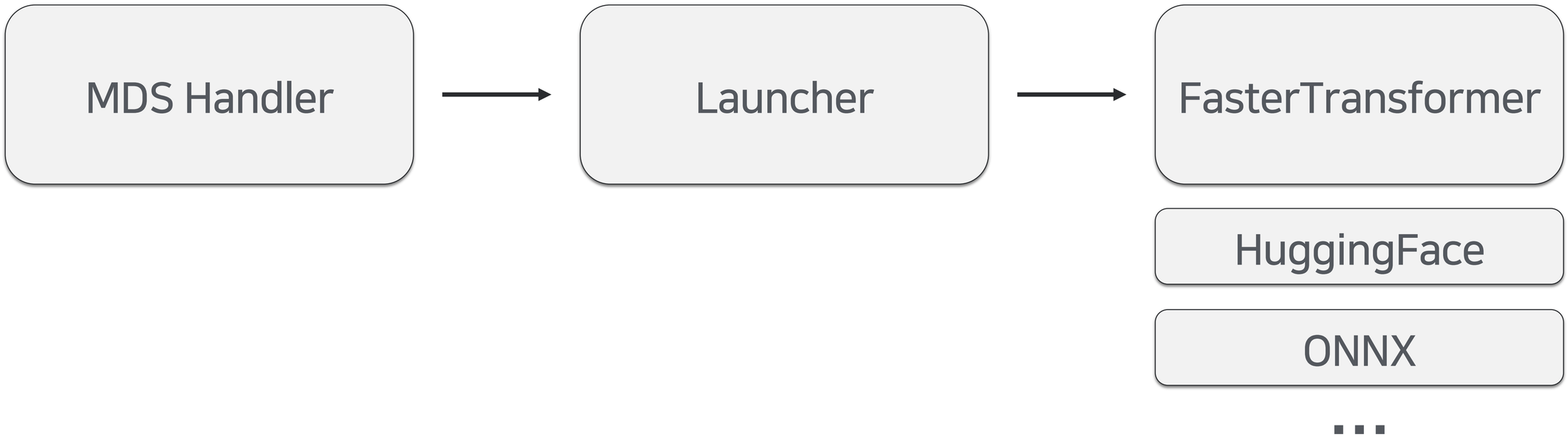
1. 생성 문장 패턴을 검색을 통한 조기 종료 지원

5. 향후 계획

5.1 Transformer 프레임워크

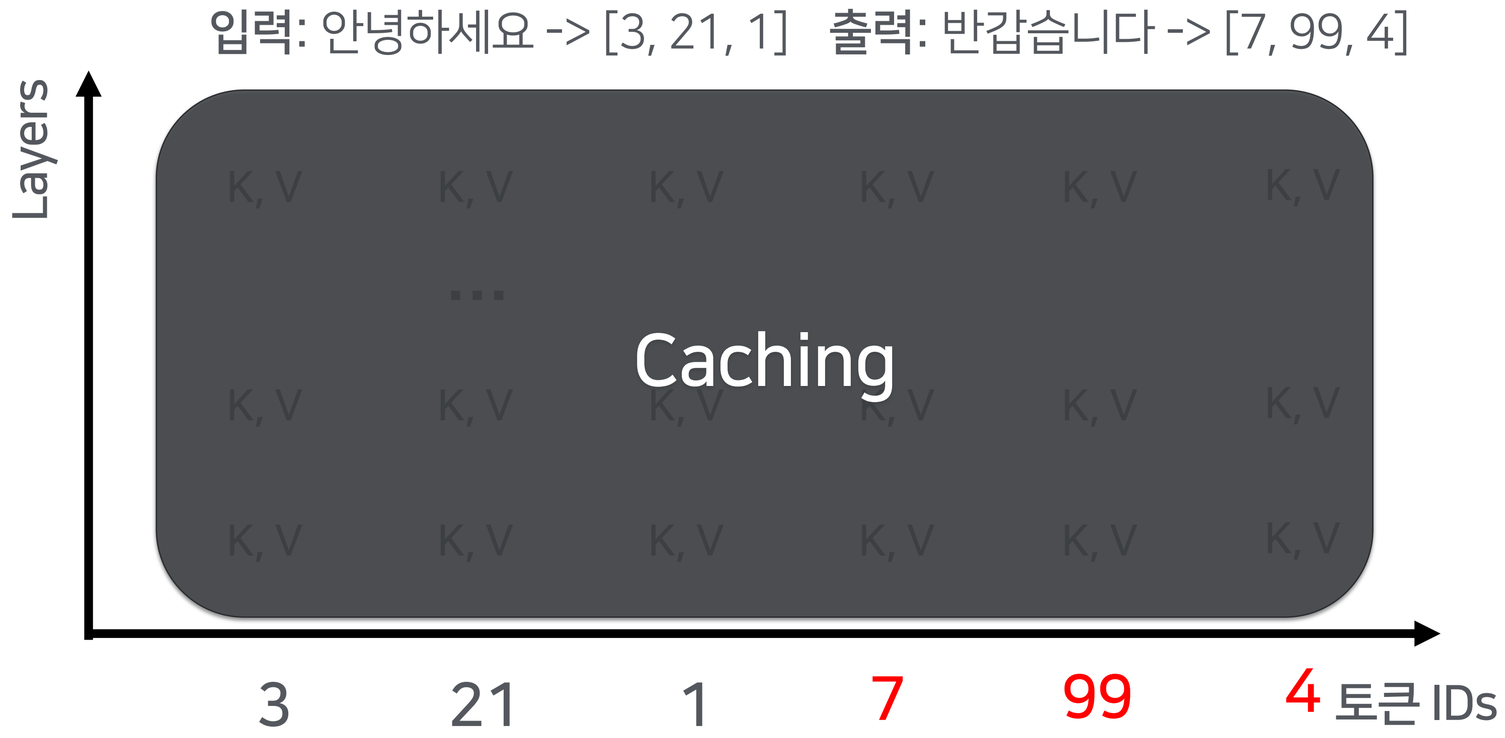
DEVIEW 2021

최적화 된 Transformer 구현체 지원



5.2 Caching 전략

Summation Stage 스킵을 위한 Key, Value 캐싱

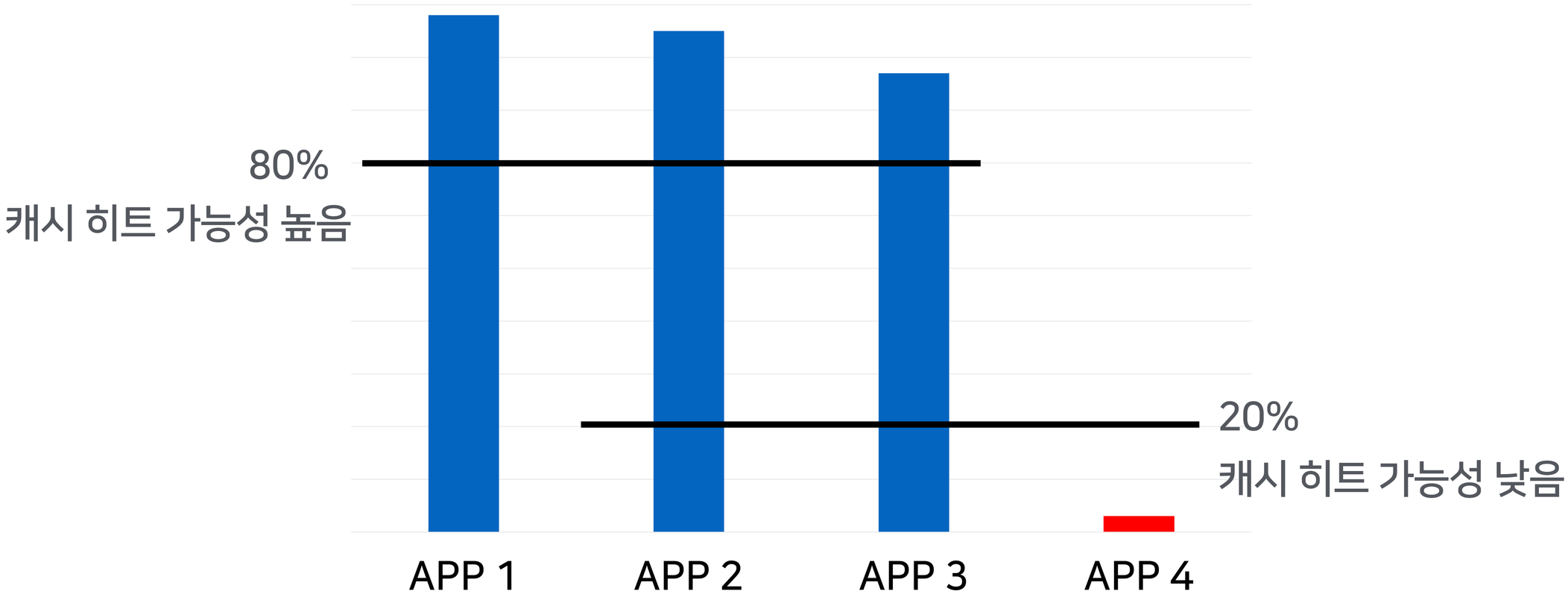


이미 연산된 문자열(입력 + 출력) 중 일부분이

처음부터 순서대로 들어올경우 연산 스킵 가능!!

5.2 Caching 전략

서비스별 일부 입력 재사용 비율



5.2 Caching 전략

문제점

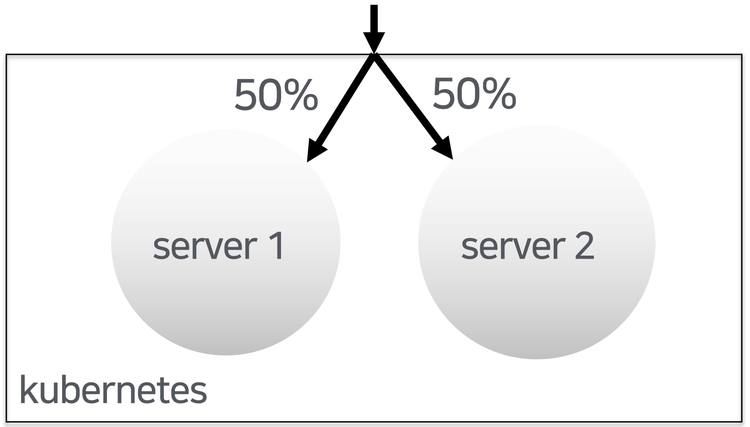
1. 사이즈

- 캐싱 대상: key, value
- 각각의 크기 : seq_len * head_nums * size_pear_head * layer_nums * sizeof(type)

⇒ 모델 크기에 따라 request 1개의 결과를 caching 할 때 **수십 GB 까지**도 늘어날 수 있음

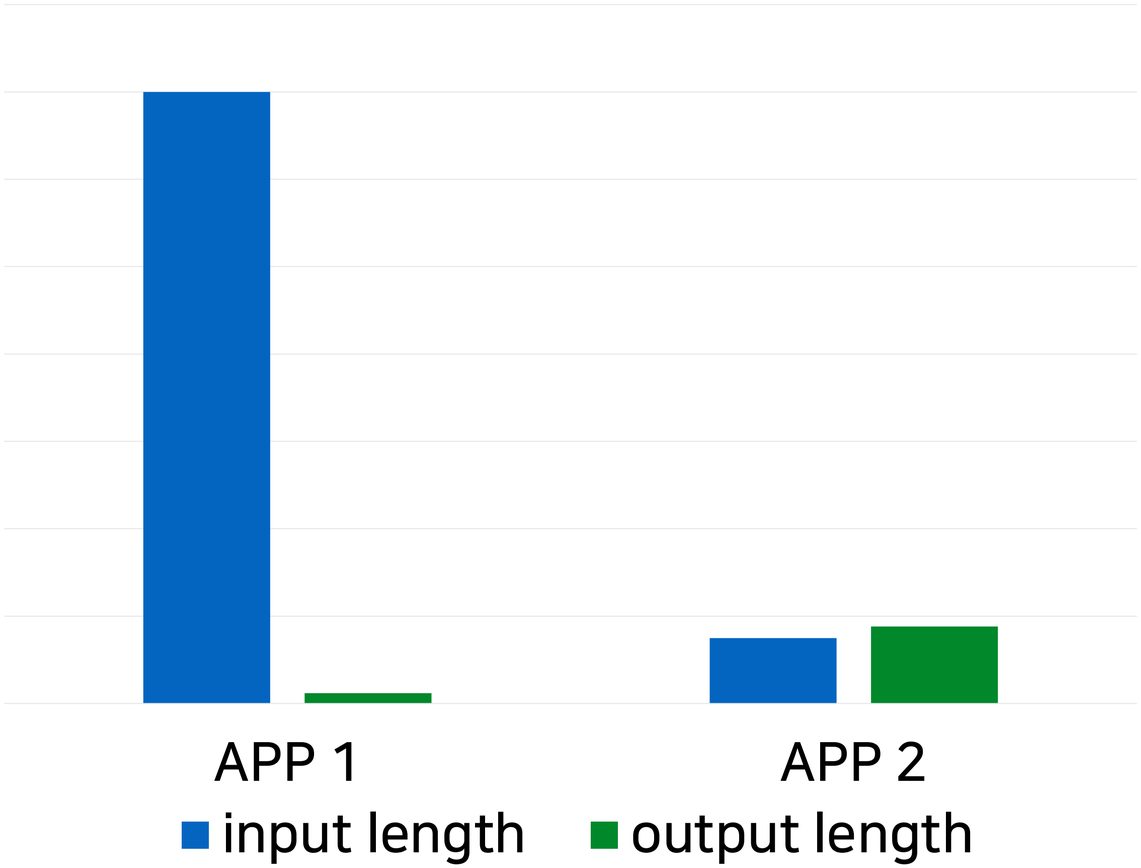
⇒ **연산 시간 < 데이터 이동 시간**

2. Caching 구조



⇒ 캐싱된 서버로 요청을 전달할 필요가 있음

5.3 최적의 하드웨어는?



- 입력 문장은 배치 처리 (summation stage)
- 문장 생성은 순차 처리 (generation stage)

- APP 1: 배치 처리가 대부분이어서 GPU가 유리
- APP 2: 순차 처리가 많아서 GPU의 경우 utilization이 낮아짐 -> FPGA, ASIC과 같은 전용 하드웨어가 유리할 가능성이 존재

Thank You