An Efficient Transfer Learning Based Configuration Adviser for Database Tuning

연세대학교 컴퓨터과학과 권세인

2024년 7월

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

정보통신기술진흥센터
Institute for Information & communications Technology Promotion

1.비효율적인 대규모 탐색 공간

-전역적으로 중요한 파라미터를 정적으로 선택하는 것은 비효율적
-기존 시스템은 파라미터의 기본 값 범위를 사용하여 비효율적

ex) MySQL의 innodb_buffer_pool_size의 기본 범위는

5MB~16EB

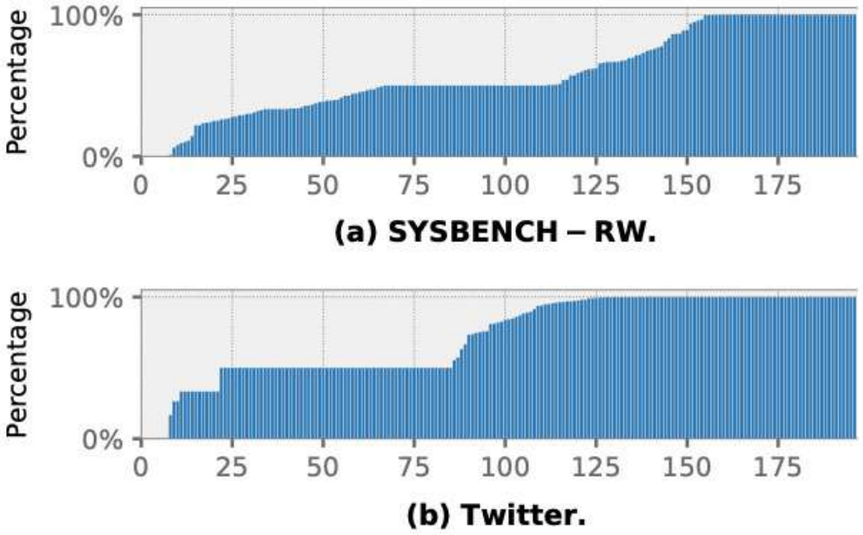


Figure 3: Default Ranges Are Superfluous – The proportion of effective range size against the default range size.

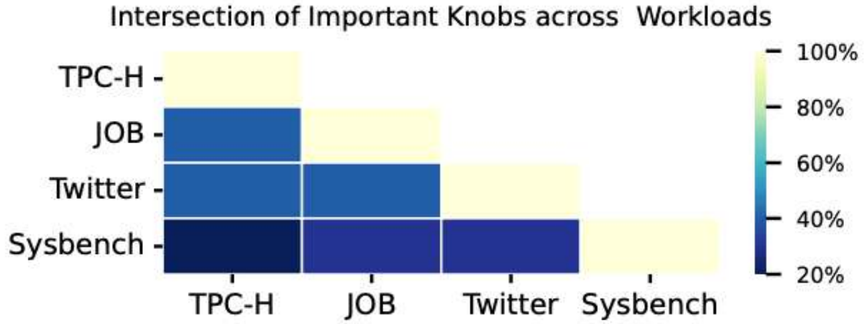


Figure 1: Important Knobs Are Workload-Specific – We present the percentage of intersection among the top-10 important knobs from four different workloads.

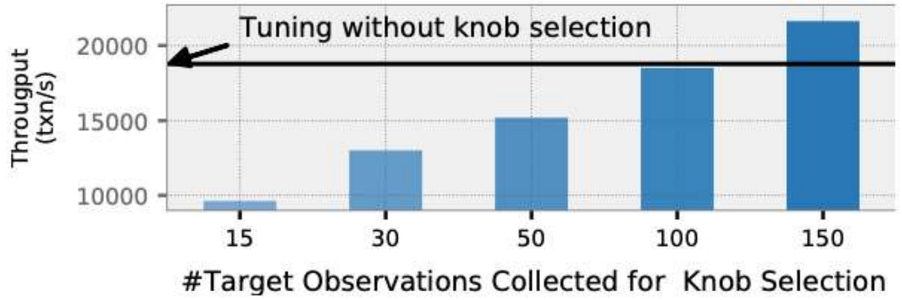


Figure 2: Well Selecting Important Knobs Takes Considerable Observations – Tuning performance on top-10 knobs selected based on different numbers of observations.

2. 단일 최적화 기법의 한계

-기존 시스템은 주로 단일 최적화 기법 사용
-특정 튜닝 작업에 적합한 탐색 최적화 기법 식별 필요

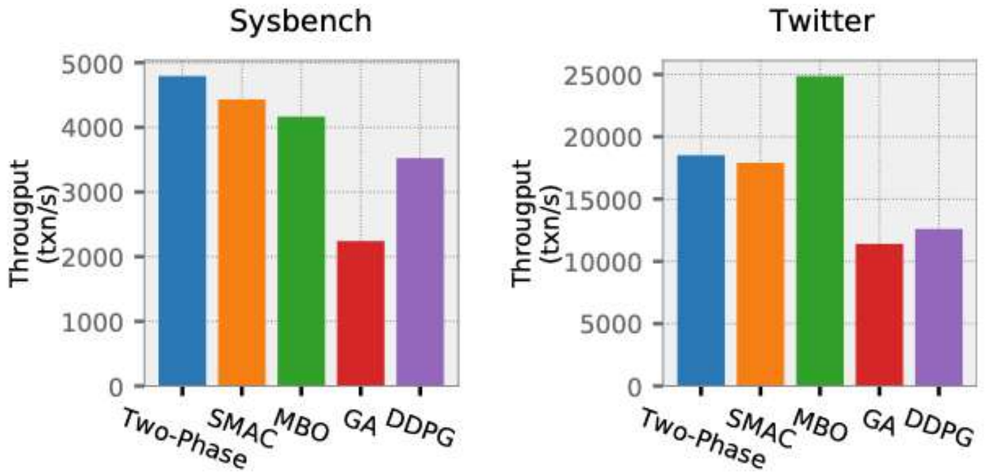


Figure 4: No Single Optimizer Dominates All Tuning Tasks – Performance of different optimizers on two workloads.

1.과거 튜닝 데이터를 활용하여 특정 튜닝 작업에 적합한 **탐색 공간 자동 구성**
2.주어진 튜닝 작업에 가장 적합한 **최적화 도구 선택**

$$\theta^* = \arg\max_{\theta \in \Theta} f(\theta). \qquad (1)$$

-DBMS는 각각의 고유한 도메인 $\Theta_1, \ldots, \Theta_n$을 가진 n개의 조정 가능한 노브 $k_1, \ldots, k_n$ 갖고 있음.
-연속 도메인: $\Phi_i = (\hat{l}_i, \hat{u}_i)$, 범주형 도메인: $\Phi_i = \hat{s}_i$
-노브 집합이 주어지면, 탐색 공간은 $\Theta = \Theta_1 \times \ldots \times \Theta_n$

-반복 t에서 채택된 탐색 최적화 기법 Ot
예)

$$O_t = \begin{cases} GA, & t \leq 140 \\ DDPG, & t > 140 \end{cases} \qquad (2)$$

-튜닝 작업 wi 의 과거 관찰을 $H^i = \{\theta^i_j, f(\theta^i_j, w_i)\}_{j=0}^{T_i}$ 로 나타냄.
-여기서 $f(\cdot, w_i)$는 튜닝 작업 wi 하에서의 성능 함수.

아키텍처

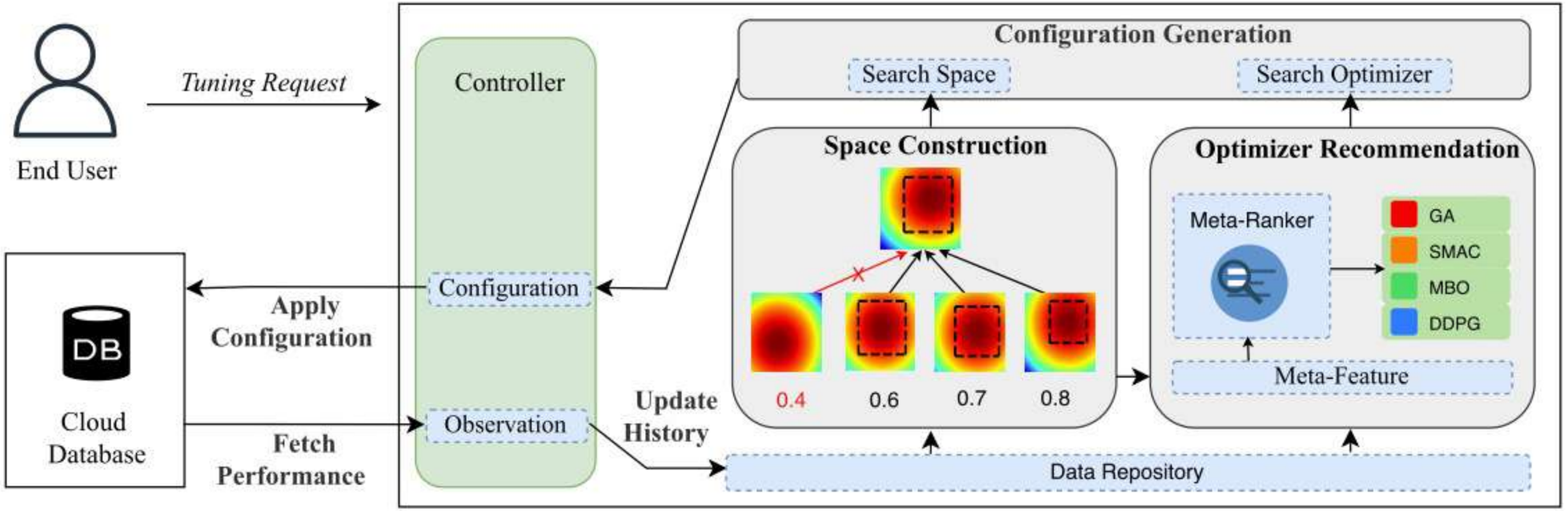


Figure 5: Overview: Architecture and Workflow of OpAdviser.

1.컨트롤러: 최종 사용자와 데이터베이스 인스턴스와 상호작용하여 튜닝 프로세스 제어

2.데이터 저장소: 튜닝 관련 데이터를 저장, 다른 튜닝 작업의 과거 관찰도 포함됨

3.탐색 공간 생성기: 컴팩트한 탐색 공간 구성

4.최적화 추천기: 적절한 탐색 최적화 기법 선택

5.구성 생성기: 선택된 최적화 기법을 사용하여 유망한 구성 생성

워크플로우

튜닝 목표, 튜닝 예산,

DB 인스턴스, 목표 워크로드

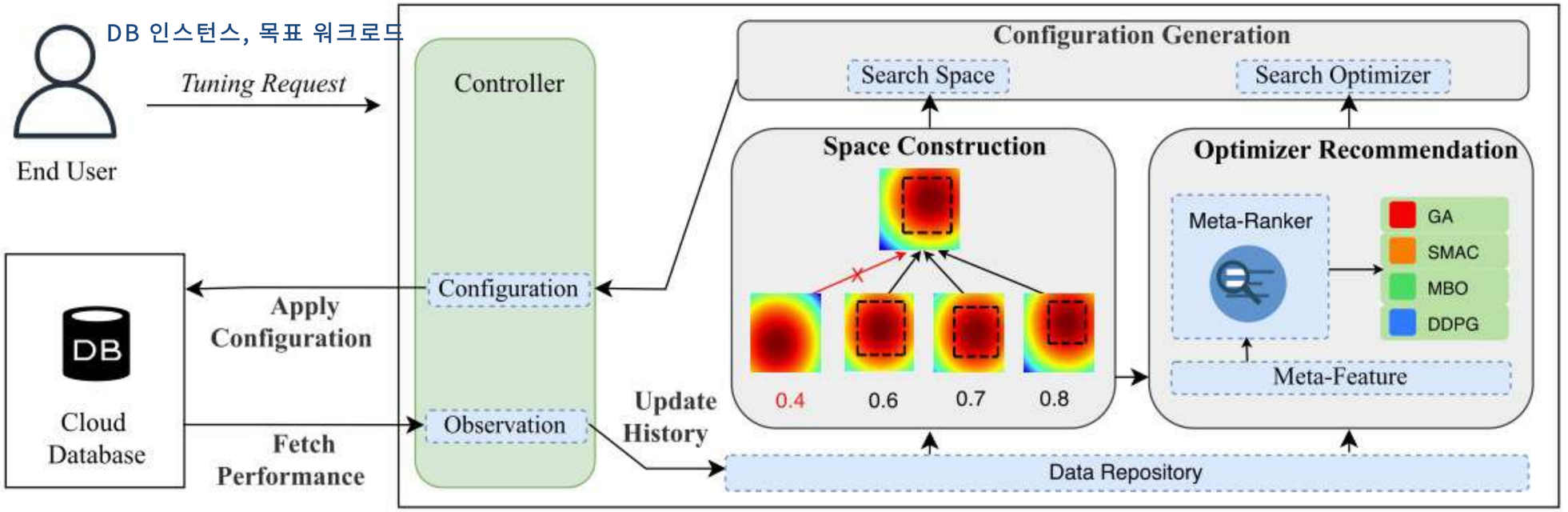


Figure 5: Overview: Architecture and Workflow of OpAdviser.

1.설정 적용 및 성능 획득

2.데이터 저장

3.탐색 공간 구성

4.최적화 기법 선택

5.구성 생성

6.반복

탐색 공간 구성-유사 작업 식별

유사한 작업 식별의 필요성

- 튜닝 작업을 수행할 때, 이전 작업에서 얻은 통찰력 활용
- 잘못된 탐색 영역 피할 수 있음

유사성 측정 방법

- 목표 관찰 값을 기반으로 한 일치하는 순위 쌍의 비율로 측정

$$S(i,t) = \frac{2}{|H^t|(|H^t|-1)} F(i,t)$$

ex) n번째 반복에서 타겟 작업의 관찰 값: θ1, θ2, θ3

- 타겟 작업의 성능 순위:
  - f(θ1) ≤ f(θ2)
  - f(θ2) ≥ f(θ3)
  - f(θ1) ≤ f(θ3)
- 소스 작업 w_3의 성능 순위:
  - f'(θ1, w3) ≤ f'(θ2, w3)
  - f'(θ2, w3) ≥ f'(θ3, w3)
  - f'(θ1, w3) ≥ f'(θ3, w3)
- 두 개의 순위 쌍이 일치:
  - f'(θ1, w3) ≤ f'(θ2, w3)
  - f'(θ2, w3) ≥ f'(θ3, w3)
- 따라서 $S(3,t) = \frac{2}{3}$

탐색 공간 구성-유효 영역 추출

Methodology:

- 목표는 유효 영역을 작고 효과적으로 유지하면서도 최적 구성을 포함하도록 하는 것

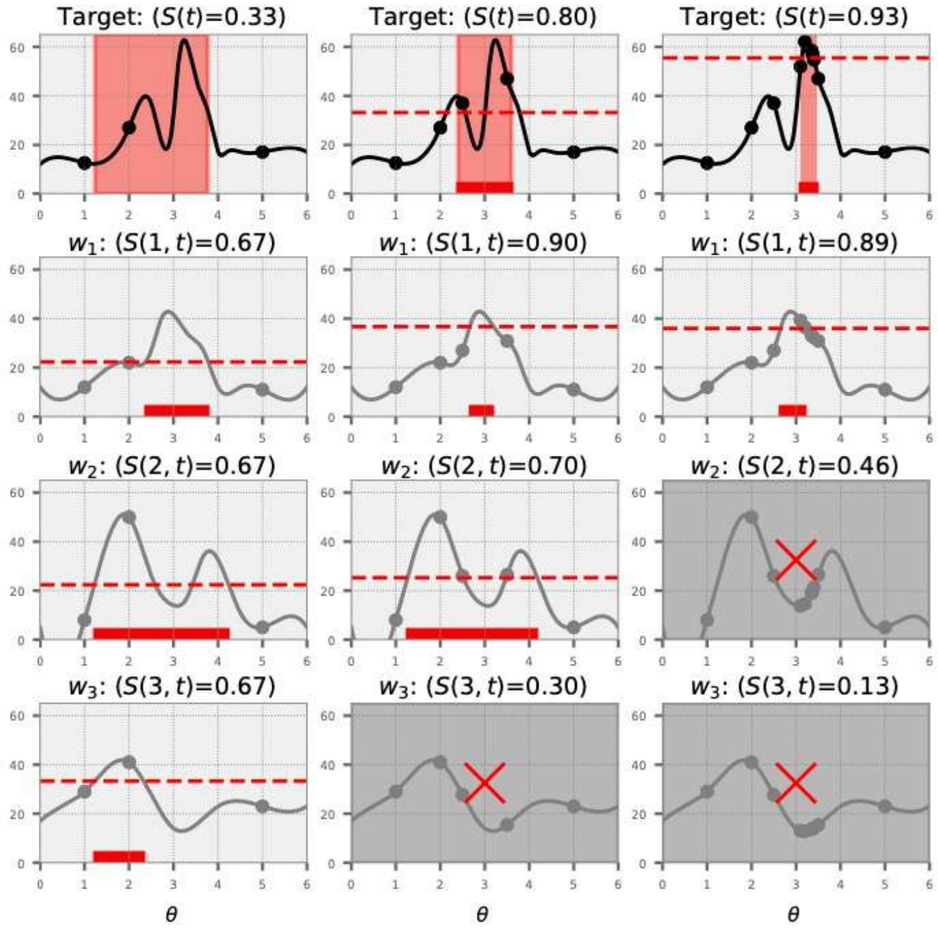
$$\begin{aligned} &\arg\min_{\Phi^i} \Lambda(\Phi^i), \\ \text{s.t.} \quad &\forall \theta \in G, \theta \in \hat{\Theta}(\Phi^i). \end{aligned} \tag{4}$$

Configurations:

- $G = \left\{\theta \in \Theta | f(\theta, w^i) \geq f_b^i\right\}$

Feature Selection:

- 과거 관찰 Hi를 기반으로 중요한 노브 식별을 위해 SHAP 사용
- 각 파라미터의 기여도를 계산하여 중요하지 않은 파라미터 제외



(a) Iteration 4. (b) Iteration 6. (c) Iteration 11.

Figure 6: A Toy Example, Showing Three Steps of Space Construction: (1) Computing similarity $S(i, t)$ based on the number of concordant ranking pairs between the target performance (black point) and the predicted performance (grey point) and filtering tasks with $S(i, t) < 0.5$ (red cross). (2) Extracting effective regions (red area) with performance better than a standard (red dashed line) controlled by $S(i, t)$. (3) Constructing target range (pale pink shading in the first row) by a weighted combination of the red areas.

탐색 공간 구성-다수 가중 투표

가중치 계산:

- 각 소스 작업 i의 유사성 S(i,t)를 기반으로 가중치 wi를 계산함. 아래 식에서 m은 소스 작업의 수.

$$w_i = \frac{S(i,t)}{\sum_{j=1}^{m} S(j,t)}$$

다수 가중 투표:

- 가중치가 적용된 다수 투표를 통해 최종 유효 영역 결정
- 투표의 총 가중치 임계값은 50%로 설정되며, 다수의 동의를 얻은 노브와 범위만 최종 유효 영역에 포함됨

부정적 전이 방지:

- 부정적 전이는 유사하지 않은 소스 작업의 정보가 목표 작업의 최적화에 부정적인 영향을 미치는 현상
- 현재 최적화 하려는 목표 작업 자체를 소스 작업들과 동일한 방식으로 하나의 투표자로 간주
- 목표 작업과 유사하지 않은 소스 작업의 영향을 줄이고, 목표 작업의 특성에 맞는 유효 영역 생성

$$S(t,t) = \frac{2}{|H^t|(|H^t|-1)} F(t,t)$$

탐색 옵티마이저 추천-메타 피쳐 추출

- **탐색 공간의 차원과 구성/ 응답 표면/ 튜닝 단계**가 중요한 요인임

(1) Space feature-튜닝 노브 수, 탐색 공간 크기, 연속 튜닝 노브 대 범주형 튜닝 노브 비율
(2) Response surface feature-현재 작업과 각 이전 튜닝 작업 간의 일치순서 쌍 비율을 포함하는 유사 벡터
(3) Tuning process feature-튜닝 과정의 현재 반복 횟수

탐색 옵티마이저 추천-오프라인 데이터 생성

- " DDPG performs well in large search spaces.” ->기존의 휴리스틱 접근 방식의 한계
- 이를 해결하기 위해 튜닝 작업 선택을 위해 머신 러닝 모델을 활용한 데이터 기반 솔루션 제안
- Active Learning 기법을 사용하여 샘플을 선택하고 라벨링 함
- 효율적인 데이터 수집을 가능하게 하며, 테스트 비용을 줄이는데 큰 도움이 됨

탐색 옵티마이저 추천-메타 랭커 구축

입력 및 훈련 데이터

- 입력: 튜닝 작업의 meta feature와 두 개의 후보 옵티마이저
- 훈련 데이터 구조: (m, o_i, o_j, I)

온라인 단계

- 실시간으로 meta feature 추출
- 후보 옵티마이저 쌍과 함께 meta ranker에 입력
- 최적의 옵티마이저 추천

실험 평가

- OpAdviser-w/o-Optimizer: 최적화 도구 추천 기능을 제거한 버전
- OpAdviser-w/o-Space: 탐색 공간 생성기를 제거한 버전

- OpAdviser가 평균적으로 3.4배 더 빠르게 최상의 구성 도달, 절반의 튜닝 예산으로 동일한 최고의 처리량 달성
- 최종 처리량을 평균 약 9.2% 향상시킴

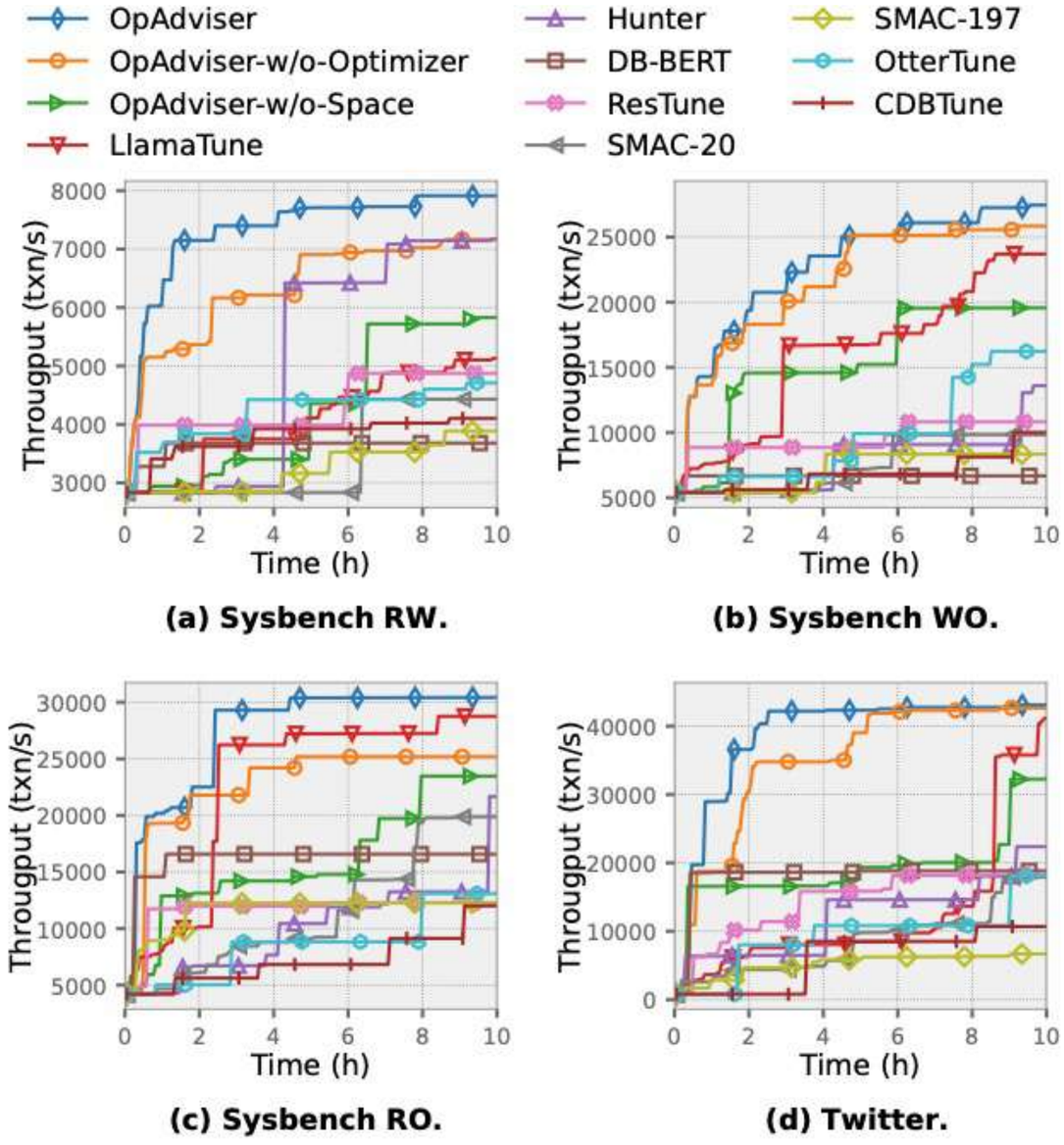


Figure 7: End-to-end Comparison over The Tuning Time.

실험 평가

- 목표 워크로드 실행 시간을 제외한 다양한 접근법이 구성 생성을 위해 필요한 시간 측정
- OpAdviser의 알고리즘 시간이 더 길더라도, 목표 워크로드 실행 횟수를 줄임

Table 1: Algorithm Time Per Iteration on Average.

OpAdviser	LlamaTune	Hunter	DB-BERT	ResTune	OtterTune	CDBTune
5.92s	1.67s	0.02s	9.61s	6.86s	6.89s	0.13s

- **Static:** 일반적인 중요도에 대한 정적 순위를 기반으로 상위 20개의 노브로 정의된 탐색 공간 구성
- **Increase:** 상위 4개의 노브로 튜닝을 시작하고 정적 순위에 따라 매 4회 반복마다 2개의 노브 추가
- **Decrease:** 모든 노브로 튜닝을 시작하고, 이전 10개의 관찰로 생성된 중요도 순위를 기반으로 매 10회 반복마다 중요하지 않은 노브의 40% 제거
- **Box:** 이전 모든 작업의 최상의 관찰된 구성을 포함하는 최소 탐색 공간 제안

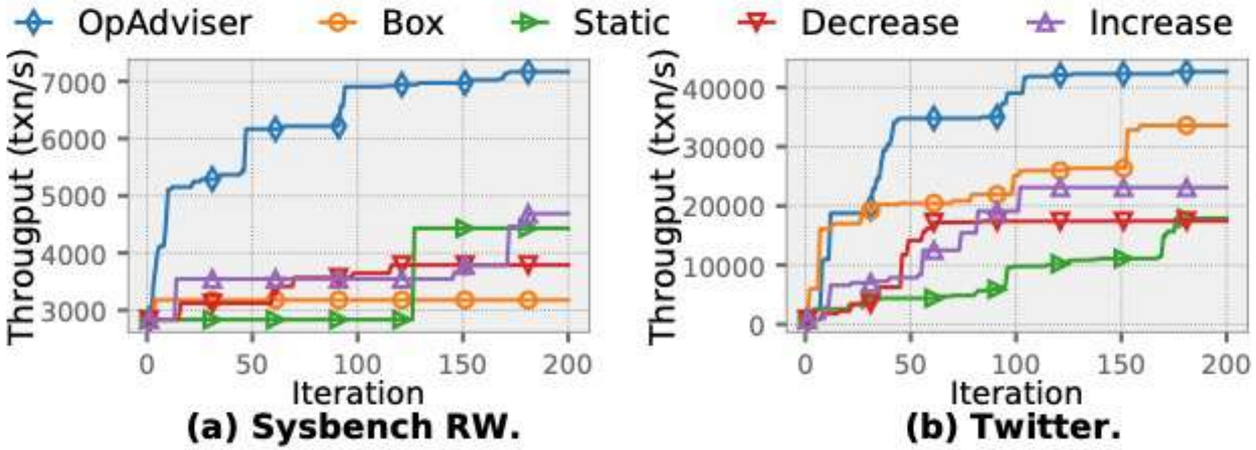


Figure 8: Comparison of Different Space Strategies.

실험 평가

- **단일 옵티마이저 사용:** SMAC, MBO, DDPG, GA와 같은 단일 옵티마이저를 사용하는 전략
- **OpAdviser:** 두 후보 옵티마이저의 상대적 성능을 예측하는 페어와이즈 랭킹 모델을 사용하는 방식
- **Classify:** 최고의 옵티마이저를 직접 예측하는 분류 모델을 사용하는 OpAdviser의 변형
- **Regress:** 후보 옵티마이저의 절대 성능을 예측하여 가장 높은 성능을 가진 옵티마이저를 추천하는 회귀 모델을 사용하는 OpAdviser의 변형
- **Two-Phase:** Hunter의 접근 방식으로, 첫 140번의 반복 동안 GA를 사용한 후 DDPG로 전환

Table 2: Ablation in Space Construction on Sysbench RO.

Module			Throughput	Improvement
Knob	Range	Similarity	(txn/sec)	ratio
-	-	-	12240	-
✓	✓	-	9742	-20.4%
✓	-	✓	15872	29.7%
-	✓	✓	21376	74.6%
✓	✓	✓	25406	107.6%

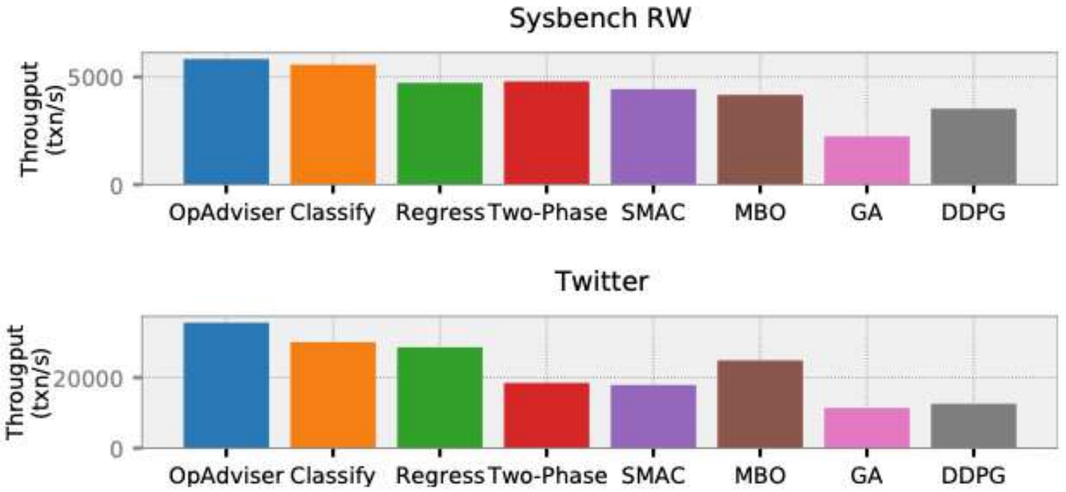


Figure 9: Comparison of Optimizer Selection Strategies.

실험 평가

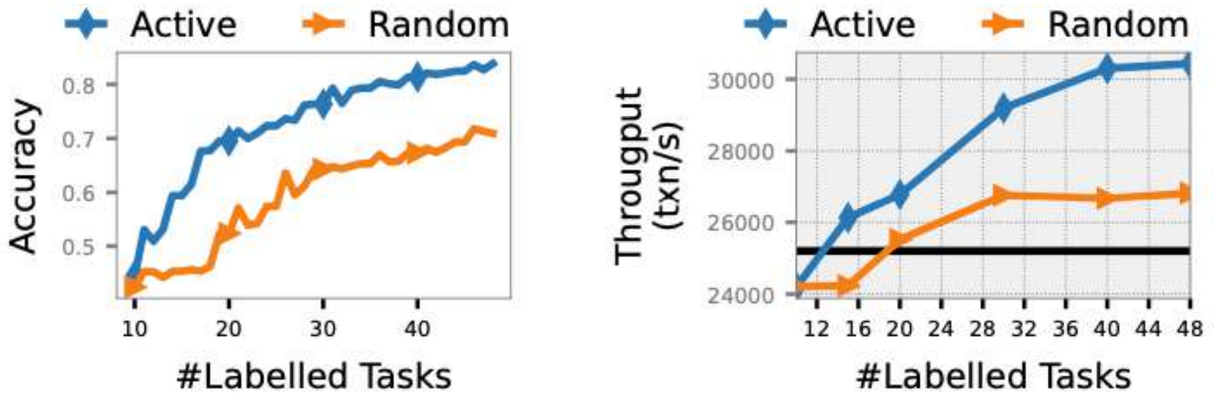


Figure 10: Utility of Active Learning. In Figure (a), we present the accuracy of the meta-ranker trained via active learning and random sampling. Figure (b) displays the associated tuning performance, with a horizontal line representing the baseline performance with SMAC as the default optimizer.

- (a): 랜덤 샘플링과 비교해서 Active Learning을 사용했을 때 최적의 옵티마이저에 대한 분류 정확도
- (b): Sysbench RO 워크로드에서의 튜닝 성능

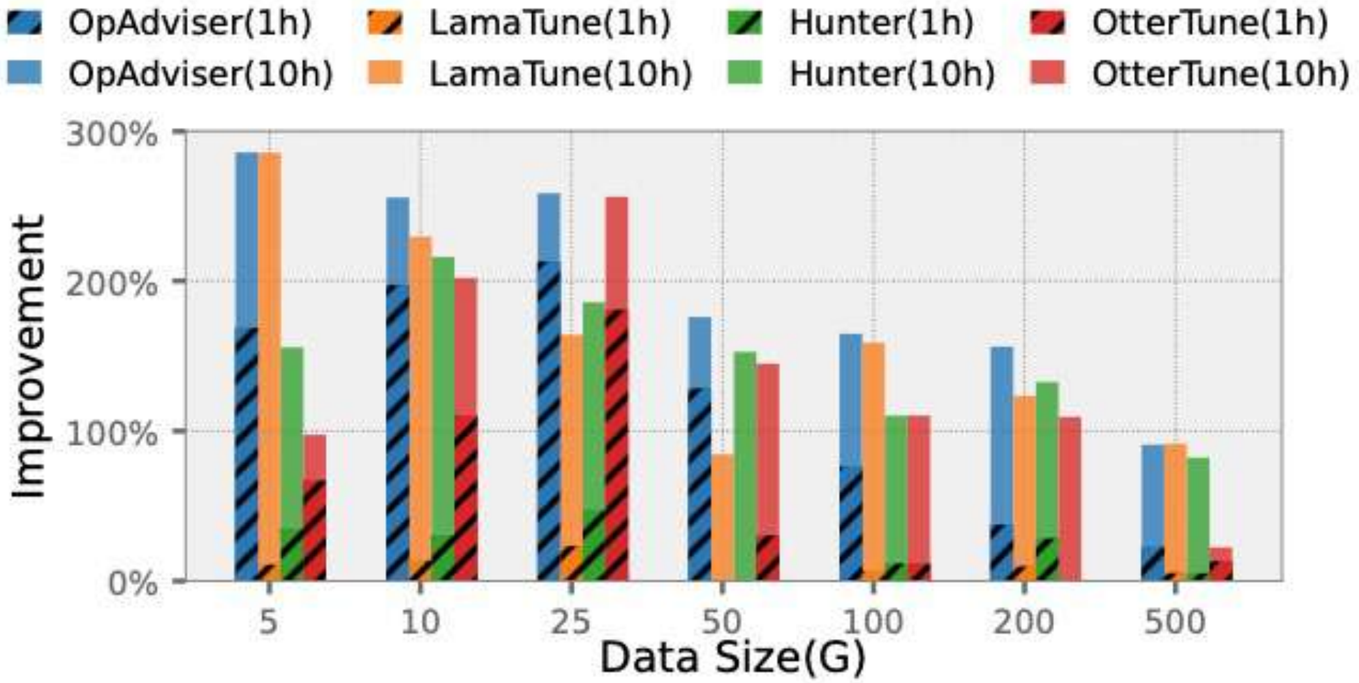


Figure 11: Generalization across Different Data Sizes.

- 다양한 데이터 크기에서의 일반화 성능
- 막대 그래프는 1시간 튜닝 예산(해치된 막대)과 10시간 튜닝 예산(솔리드 막대) 내에서의 성능 향상 나타냄

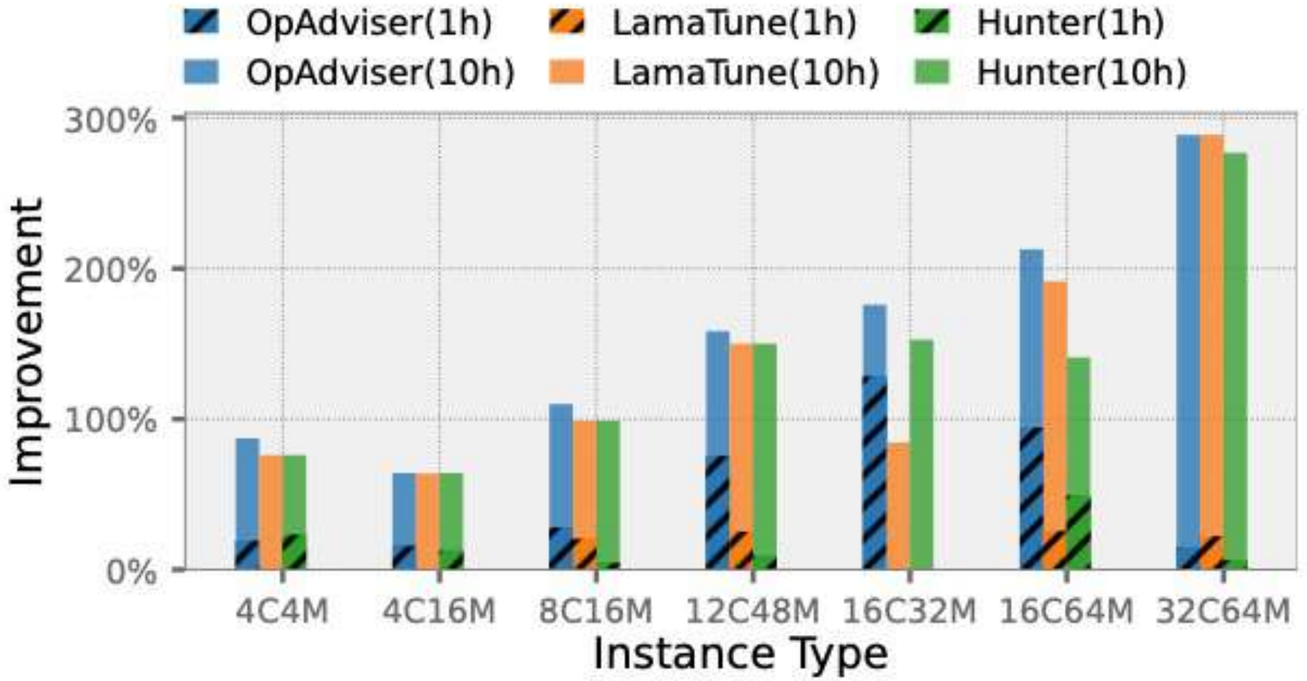


Figure 12: Generalization across Hardware Environments.

- 다양한 하드웨어 환경에서 일반화 성능
- 인스턴스 유형은 “4C16M” 처럼 표기되고, 여기서 “4C”는 4개의 CPU를, “16M”은 16GB 메모리 의미

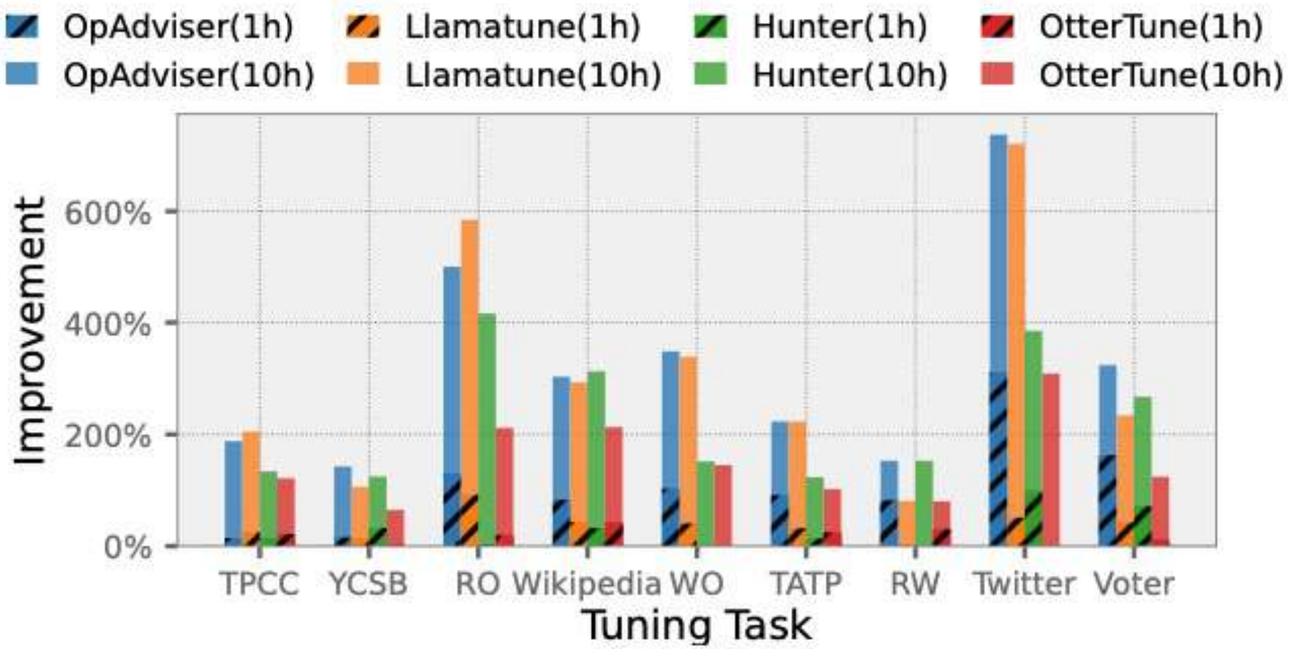


Figure 13: Evaluation on Cold Start Scenario: OpAdviser's advantage increases as completing more tuning tasks.

- 초기의 과거 데이터 없이도 얼마나 효과적으로 성능을 발휘하는지 평가한 결과
- 공정성을 위해 최적화 도구 추천 모듈을 비활성화 하고, SMAC을 최적화 도구로 채택

이전 튜닝 시스템의 한계

1. 비효율적인 대규모 탐색 공간

- 중요한 파라미터 찾기 위해 많은 워크로드 실행 필요
- 기본 값 범위가 너무 넓어 비효율적 탐색 발생

2. 단일 최적화 기법의 한계

- 모든 튜닝 작업에 단일 최적화 기법 적용
- 최적화 기법이 다양한 특성을 가진 작업에서 항상 우수한 성능 발휘하지 못함

OpAdviser의 해결 방법

1. 컴팩트한 탐색 공간 구축

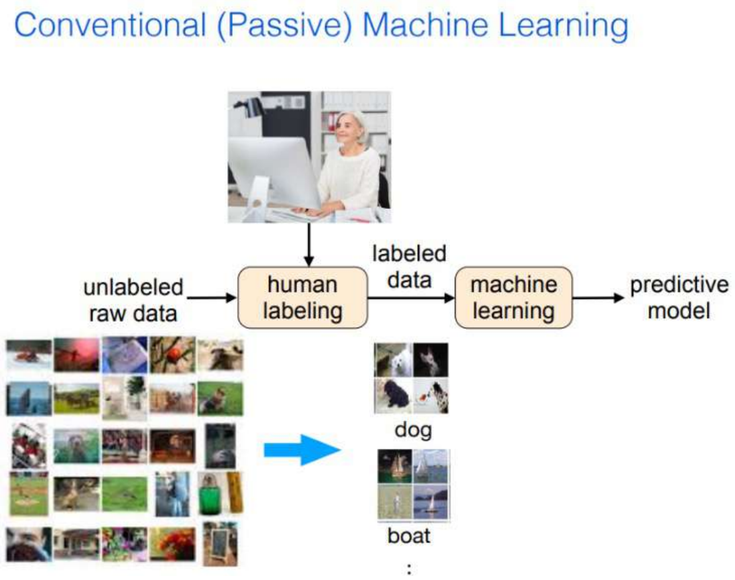
- 유사 작업의 관찰 데이터를 활용하여 탐색 공간 축소
- 중요한 매개변수와 그 값 범위 동적 조정

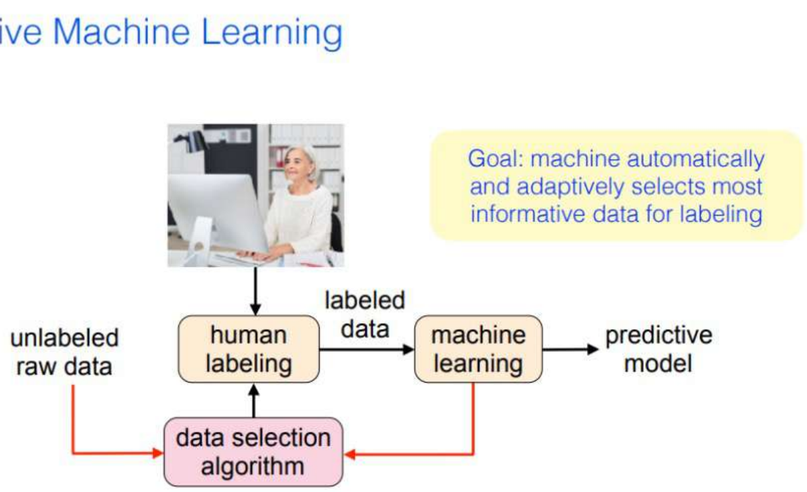
2. 다양한 최적화 기법 추천

- 메타-랭커를 통해 특정 작업에 적합한 최적화 기법 추천
- 다양한 특성을 가진 작업에 대해 높은 성능 발휘

Active Learning

Active Learning: 모델이 학습할 데이터 포인트를 능동적으로 선택하는 방법. 액티브 러닝의 주요 목표는 최소한의 라벨링으로 최대한의 성능을 얻는 것





쿼리 전략: 모델이 어떤 데이터를 학습할지 선택하는 기준.

- **불확실성 샘플링:** 모델이 가장 불확실해 하는 데이터 포인트 선택.
  ex) 확률이 가장 낮은 클래스를 예측하는 데이터 포인트 선택
- **쿼리-보수:** 여러 모델의 집합이 의견이 가장 일치하지 않는 데이터 포인트 선택

1. 데이터 수집
2. 초기 모델 학습
3. 라벨링되지 않은 데이터 중 선택
4. 라벨링
5. 모델 재학습
6. 반복

감사합니다