

아쿠아포닉스 시스템을 위한 Edge AI 활용 가이드

Edge AI란 클라우드 서버가 아닌 로컬 장치(센서, 마이크로컨트롤러 등)에서 직접 AI 알고리즘을 실행하는 기술입니다.

목차

1. Edge AI 개요
2. if문 기반 제어 vs AI 기반 제어
3. 아쿠아포닉스에서 AI가 할 수 있는 역할
4. if문으로 대체 불가능한 진정한 AI 기능
5. 구현 가능한 Edge AI 기능
6. 하드웨어 요구사항
7. 개발 도구 및 프레임워크
8. 참고 자료

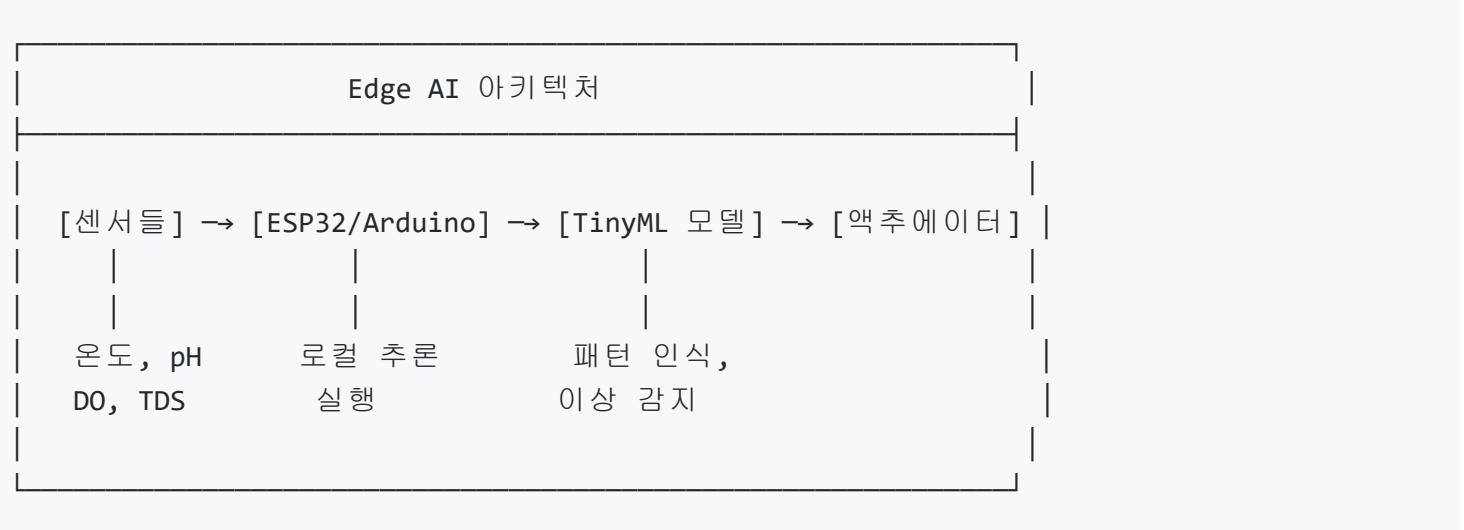
1. Edge AI 개요

Edge AI가 무엇인가요?

Edge AI는 데이터가 생성되는 곳(엣지)에서 바로 AI 추론을 수행하는 기술입니다. 아쿠아포닉스 시스템에서는 ESP32나 Arduino 같은 마이크로컨트롤러에서 직접 머신러닝 모델을 실행합니다.

TinyML이란?

TinyML은 마이크로컨트롤러와 같은 저전력, 제한된 메모리 환경에서 머신러닝을 실행하는 기술입니다.

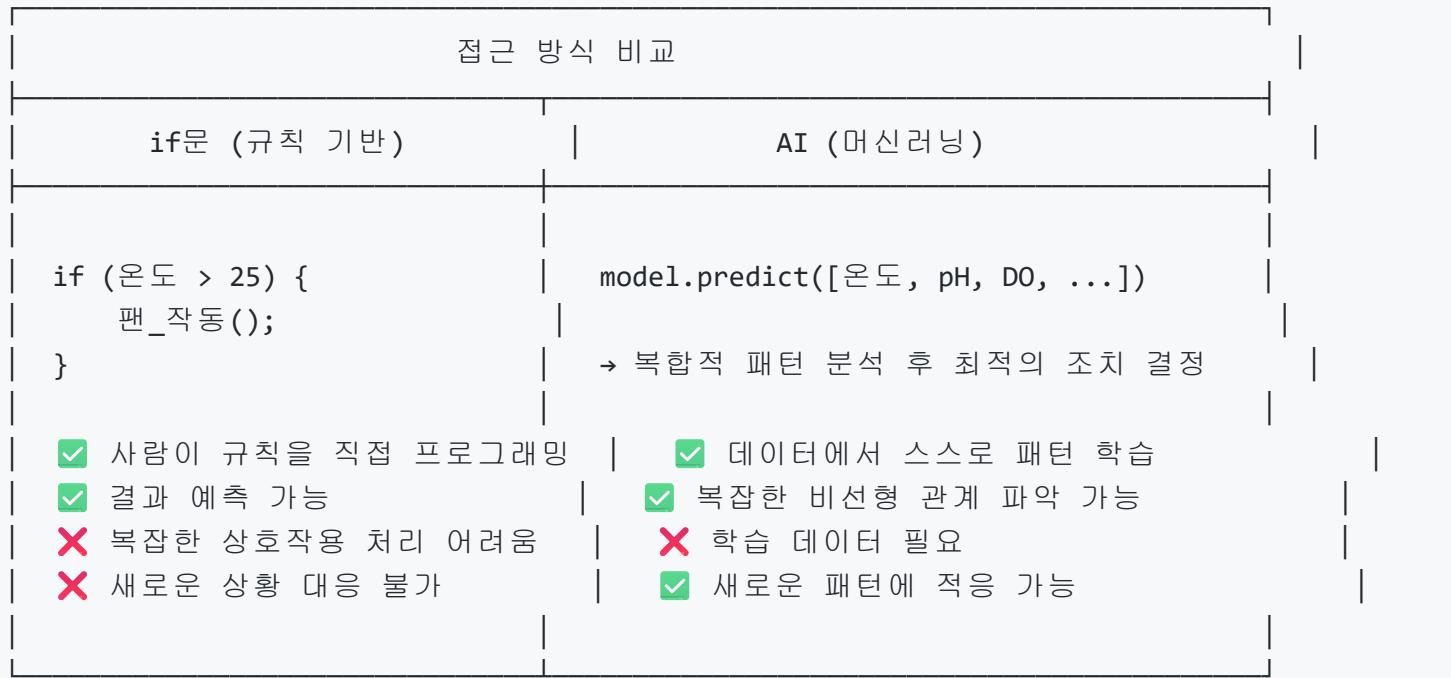


Edge AI의 장점

특성	장점
실시간 처리	네트워크 지연 없이 즉각적인 응답
프라이버시	데이터가 로컬에 유지됨
오프라인 작동	인터넷 연결 불필요
저전력	클라우드 통신 없이 낮은 에너지 소비
비용 절감	클라우드 서비스 비용 없음

2. if문 기반 제어 vs AI 기반 제어

핵심 차이점



상세 비교표

비교 항목	if문 (규칙 기반)	AI (머신러닝)
규칙 생성	사람이 직접 프로그래밍	데이터에서 자동 학습
복잡도 처리	단순 조건만 가능	복잡한 다변수 관계 처리
유연성	정적, 변경 시 직접 수정 필요	동적, 새 데이터로 적응
설명 가능성	높음 (규칙이 명시적)	낮을 수 있음 (블랙박스)
개발 비용	낮음	학습 데이터 수집 필요
패턴 인식	불가능	가능
예측 능력	없음	미래 상태 예측 가능
이상 탐지	임계값 기반만 가능	정상 패턴 학습 후 편차 감지

if문의 한계 예시

// if문으로는 이런 복잡한 판단이 어렵습니다:

// **X** 한계 1: 복합 조건의 상호작용

// "온도 23°C, pH 7.1, DO 6.5mg/L, 암모니아 0.02ppm일 때

// 4시간 후 물고기 스트레스가 발생할까?"

// → 단순 if문으로 표현 불가

// **X** 한계 2: 패턴 기반 판단

// "지난 3일간의 수질 변화 패턴을 보았을 때

// 오늘 오후에 조류 번식이 시작될까?"

// → 시계열 패턴 분석 필요

// **X** 한계 3: 개체별 적응

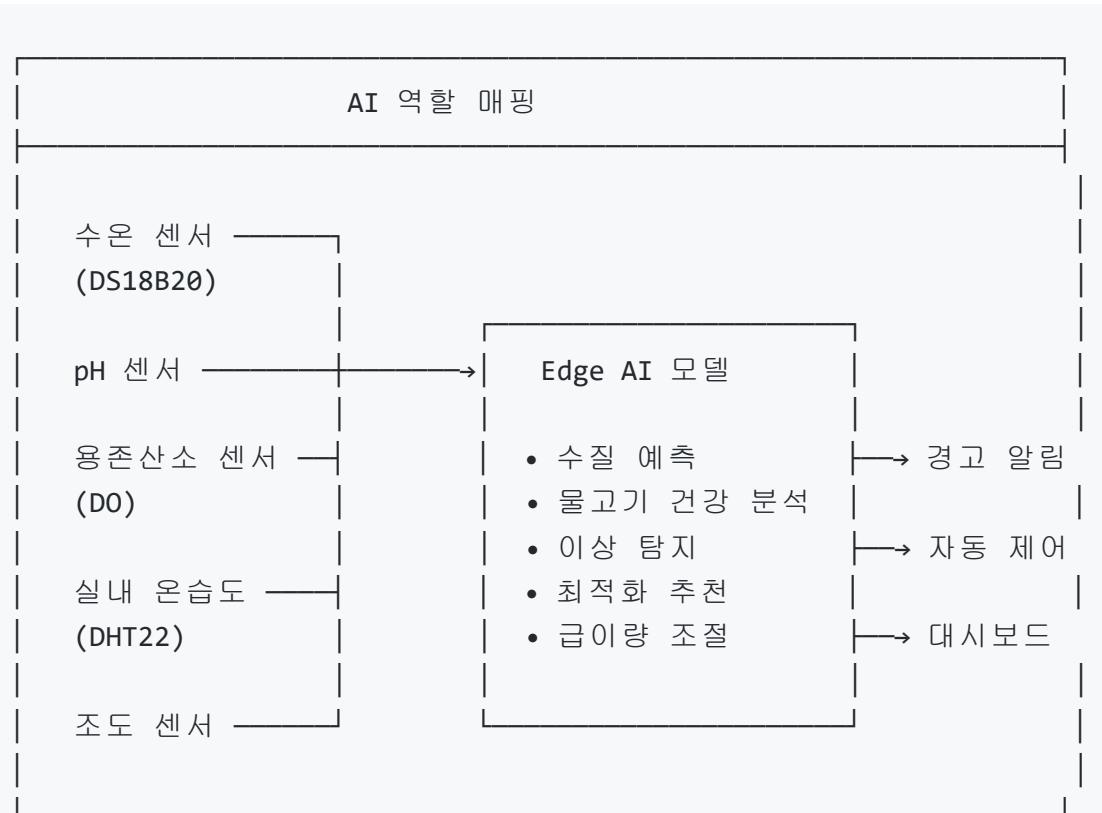
// "우리 시스템의 금붕어들은 일반적인 기준보다

// 약간 높은 온도를 선호하는 것 같은데?"

// → 시스템 특성 학습 필요

3. 아쿠아포닉스에서 AI가 할 수 있는 역할

3.1 센서 데이터 기반 역할



3.2 AI가 제공할 수 있는 실질적 도움

A. 수질 관리

기능	설명	실질적 도움
수질 예측	향후 pH, 암모니아 수준 예측	문제 발생 전 선제적 대응 가능
물갈이 시점 추천	최적의 부분 환수 시기 제안	불필요한 환수 방지, 물 절약
영양소 균형 분석	질소 순환 상태 분석	시스템 안정성 향상

B. 물고기 건강 관리

기능	설명	실질적 도움
스트레스 조기 감지	환경 지표 종합 분석	폐사 방지, 치료 시간 확보
질병 예측	이상 패턴 사전 감지	전염 방지, 의료비 절감
급이량 최적화	환경에 따른 급이량 조절	수질 오염 방지, 사료비 절감

C. 식물 생장 관리

기능	설명	실질적 도움
생장 예측	수확 시기 예측	효율적인 작물 계획
영양 결핍 경고	초기 증상 감지	수확량 감소 방지
조명 스케줄 최적화	계절/날씨 기반 조절	전력 절약, 최적 생장

D. 시스템 유지보수

기능	설명	실질적 도움
펌프 고장 예측	진동/소음 패턴 분석	갑작스러운 고장 방지
센서 드리프트 감지	센서 정확도 모니터링	잘못된 측정 방지
에너지 최적화	사용 패턴 분석	전기료 절감

4. if문으로 대체 불가능한 진정한 AI 기능

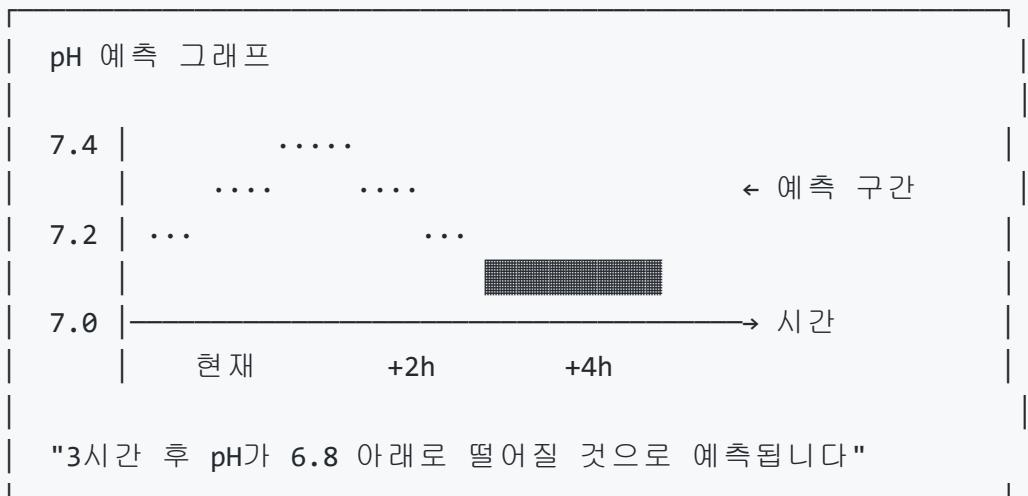
[!IMPORTANT] 이 섹션은 단순 임계값 비교로는 절대 구현할 수 없는, 진정한 AI의 가치를 보여주는 기능들입니다.

4.1 🌟 시계열 예측 (Time Series Forecasting)

왜 if문으로 불가능한가? - 과거 데이터의 복잡한 패턴과 추세를 학습해야 함 - 계절성, 주기성, 비선형 관계 등 복합 요소 분석 필요 - 미래 상태를 수학적으로 예측해야 함

[과거 24시간 pH 데이터] → [LSTM/GRU 모델] → [향후 4시간 pH 예측]

↓



구현 예시:

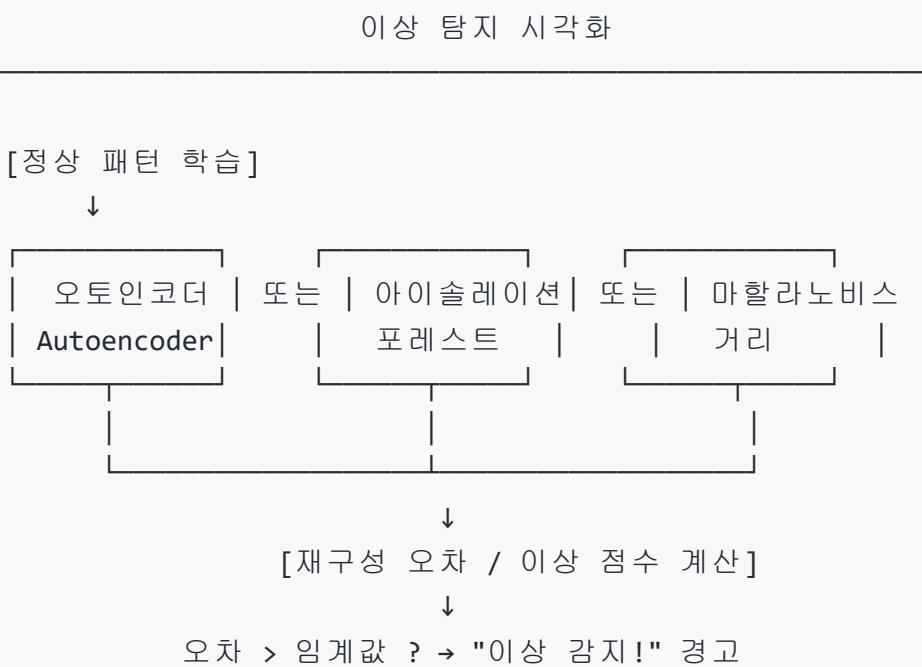
```
# TensorFlow Lite Micro를 사용한 시계열 예측
# ESP32에서 실행 가능한 경량 LSTM 모델

class WaterQualityPredictor:
    def __init__(self, model_path):
        self.interpreter = tflite.Interpreter(model_path)

    def predict_next_hours(self, last_24h_data, hours_ahead=4):
        # 입력: 지난 24시간 데이터 [pH, 온도, DO, ...]
        # 출력: 향후 n시간 예측값
        prediction = self.interpreter.invoke(last_24h_data)
        return prediction # 향후 pH, DO 등 예측값
```

4.2 🔎 이상 탐지 (Anomaly Detection)

왜 if문으로 불가능한가? - "정상"의 정의가 시간, 계절, 시스템 상태에 따라 다름 - 단일 변수가 아닌 다변수 간의 관계 이상을 감지해야 함 - 학습된 정상 패턴과의 편차를 계산해야 함



예시 :

- pH 7.0, 온도 22°C → 각각 정상 범위
- BUT: 이 조합이 이 시간대에 발생한 것은 비정상 (새벽 3시에?)
- AI가 "조합의 이상함"을 감지

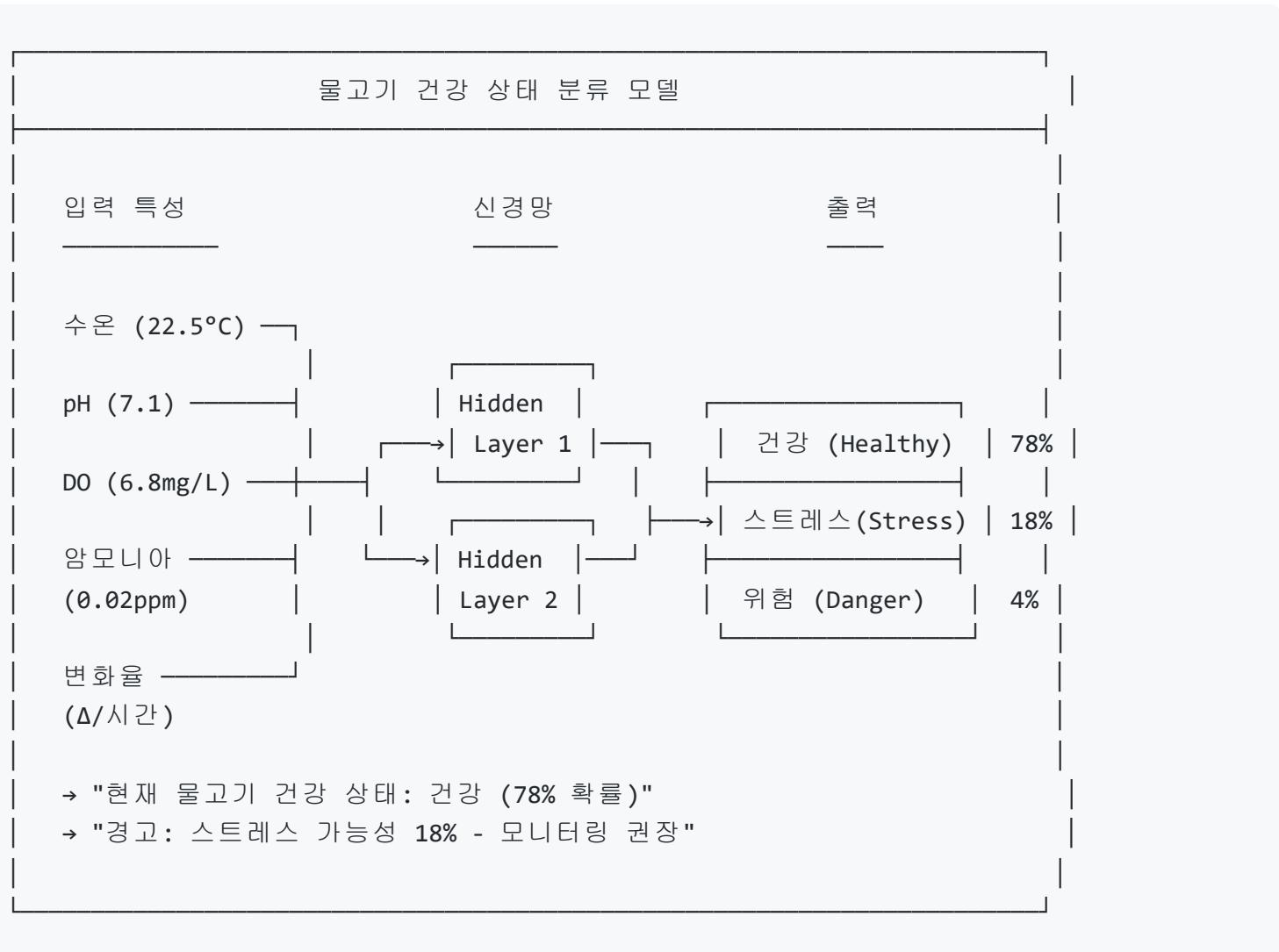
if문 vs AI 비교:

```
// ✗ if문 방식 - 한계가 명확함
if (pH < 6.5 || pH > 7.5) alert("pH 이상");
if (temp > 26) alert("온도 이상");
// 문제: pH 7.0, 온도 24°C이지만 갑자기 변화가 생겼다면?
```

```
// ✓ AI 방식 - 패턴 기반 감지
float anomaly_score = detectAnomaly([pH, temp, DO, time, ...]);
if (anomaly_score > THRESHOLD) {
    // "현재 상태는 개별 값은 정상이지만,
    // 최근 변화 패턴이 비정상입니다"
    alert("패턴 이상 감지");
}
```

4.3 물고기 건강 상태 분류 (Multi-class Classification)

왜 if문으로 불가능한가? - 여러 변수의 복합적 영향을 학습해야 함 - 건강/스트레스/위험 등 다중 클래스 분류 - 선형적이지 않은 경계 결정



4.4 다변수 상관관계 분석

왜 if문으로 불가능한가? - 5개 이상의 변수 간 복잡한 비선형 관계 - 변수들 사이의 숨겨진 상관관계 발견 - 인과 관계 추론

수온 ↔ 용존산소 ↔ pH ↔ 암모니아 ↔ 먹이량 ↔ 광량 ↔ ...

AI가 발견할 수 있는 상관관계 예시 :

"수온이 2°C 상승하고 광량이 20% 증가하면,
48시간 후 조류 성장률이 150% 증가하고
이로 인해 pH가 0.3 상승하며
물고기 스트레스 확률이 35% 증가합니다"

→ 이런 복합 관계는 if문으로 표현할 수 없음

→ 수백 개의 if문을 작성해도 불완전함

4.5 ⚡ 최적 파라미터 추천 (Optimization)

왜 if문으로 불가능한가? - 다목적 최적화 (물고기 건강 + 식물 생장 + 에너지 효율) - 제약 조건 하에서의 최적화 탐색 - 시스템 특성에 맞는 개인화

AI 최적화 추천 시스템

현재 상태

AI 추천

시스템 목표:

- 물고기 건강 우선 (가중치 50%)
- 상추 생장 (가중치 30%)
- 에너지 절약 (가중치 20%)

제약 조건:

- pH 6.8~7.2 유지
- 전기료 월 3만원 이하

최적 설정 제안:

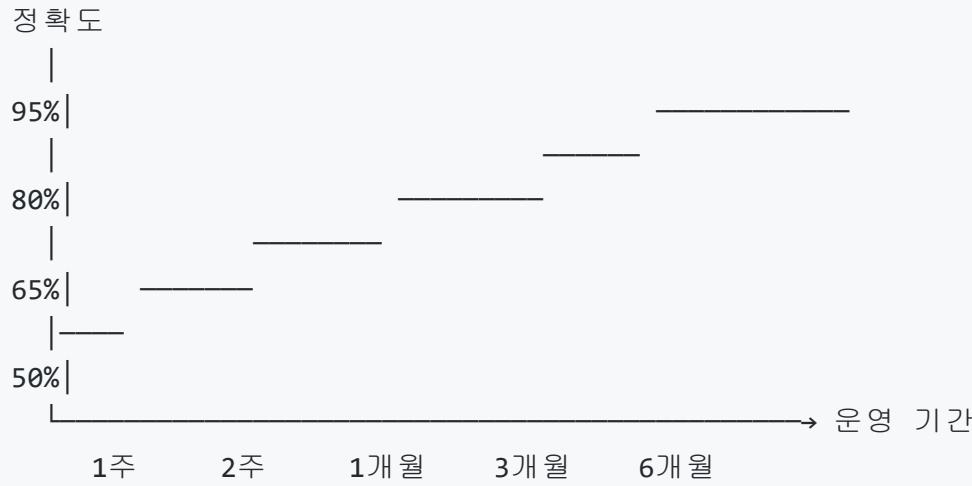
- 목표 수온: 21.5°C (현재 22.0°C)
- 펌프 사이클: 15분 온, 10분 오프
- 조명 시간: 14시간 (현재 16시간)
- 예상 결과:
 - 물고기 건강 +5%
 - 상추 생장 +3%
 - 전기 사용량 -12%

→ "이 설정으로 변경하면 현재보다 전체 효율이 8% 향상됩니다"

4.6 📈 적응형 학습 (Adaptive Learning)

왜 if문으로 불가능한가? - 시스템이 시간이 지남에 따라 변화함 - 사용자 피드백을 반영하여 모델 개선 - 개별 시스템의 특성 학습

시간 경과에 따른 AI 적응 :



"우리 시스템의 금붕어는 표준 기준 ($20-24^{\circ}\text{C}$)보다

약간 높은 온도 ($23-26^{\circ}\text{C}$)에서 더 활발합니다"

→ AI가 이를 학습하고 기준 조정

5. 구현 가능한 Edge AI 기능

5.1 난이도별 구현 로드맵

Edge AI 구현 로드맵

[초급] ━━━━━━

- 1개월 ━ 센서 데이터 수집 및 저장 시스템 구축
- ━ 기본 if문 기반 제어 완성
- ━ 데이터 시각화

[중급] ━━━━━━

- 3개월 ━ Edge Impulse로 이상 탐지 모델 학습
- ━ TensorFlow Lite Micro 모델 배포
- ━ 기본 예측 모델 구현

[고급] ━━━━━━

- 6개월 ━ 시계열 예측 (LSTM/GRU)
- ━ 다중 클래스 분류 (물고기 건강 상태)
- ━ 온라인 학습 및 적응형 모델

5.2 ESP32에서 구현 가능한 기능

기능	난이도	메모리 요구	정확도 기대치
이상 탐지 (Autoencoder)	★★☆	~50KB	85-90%
물고기 건강 분류	★★★	~100KB	80-85%
수질 예측 (4시간)	★★★☆	~150KB	75-85%
급이량 최적화	★★☆	~30KB	80-90%
센서 드리프트 감지	★★☆	~40KB	90-95%

5.3 구체적 구현 예시: 이상 탐지 모델

```

// ESP32에서 TensorFlow Lite Micro를 사용한 이상 탐지

#include <TensorFlowLite_ESP32.h>
#include "anomaly_detection_model.h"

// 모델 관련 변수
tfLite::MicroInterpreter* interpreter;
const int kInputSize = 6; // pH, 온도, D0, 암모니아, 조도, 습도

// 이상 탐지 함수
float detectAnomaly(float* sensor_data) {
    // 입력 데이터 정규화
    for (int i = 0; i < kInputSize; i++) {
        input->data.f[i] = normalize(sensor_data[i], i);
    }

    // 모델 추론 실행
    interpreter->Invoke();

    // 오토인코더: 재구성 오차 계산
    float reconstruction_error = 0;
    for (int i = 0; i < kInputSize; i++) {
        float diff = input->data.f[i] - output->data.f[i];
        reconstruction_error += diff * diff;
    }

    return sqrt(reconstruction_error / kInputSize);
}

void loop() {
    float sensor_data[6];
    readAllSensors(sensor_data);

    float anomaly_score = detectAnomaly(sensor_data);

    if (anomaly_score > ANOMALY_THRESHOLD) {
        // 이상 상황 - 경고 발생
        sendAlert("이상 패턴 감지: 점수 " + String(anomaly_score));
    }

    delay(60000); // 1분 간격
}

```

5.4 데이터 수집 전략

효과적인 AI 모델을 위한 데이터 수집:

수집 항목	샘플링 주기	최소 수집 기간	용도
수온	1분	1개월	모든 모델
pH	5분	1개월	수질 예측
용존산소 (DO)	5분	1개월	물고기 건강
암모니아	1시간	2개월	질소순환 분석
실내 온습도	5분	1개월	환경 상관관계
조도	10분	1개월	식물 생장
펌프 작동 상태	이벤트	1개월	시스템 분석
사용자 이벤트 (급이, 환수 등)	이벤트	3개월	라벨링

데이터 라벨링 예시:

- 물고기 폐사 발생 → 이전 48시간 데이터를 "위험" 라벨
- 성공적인 수확 → 해당 기간 데이터를 "최적" 라벨

6. 하드웨어 요구사항

6.1 Edge AI를 위한 추천 보드

보드	AI 성능	메모리	가격대	추천 용도
ESP32	기본	520KB SRAM	5,000원	입문, 소형 모델
ESP32-S3	향상	512KB SRAM	10,000원	벡터 연산 최적화
Arduino Nano 33 BLE Sense	좋음	256KB SRAM	35,000원	내장 센서 활용
Raspberry Pi Pico	기본	264KB SRAM	6,000원	저비용
Raspberry Pi 4	매우 높음	1-8GB	50,000원+	복잡한 모델

6.2 센서 업그레이드 제안

기존 구성에서 Edge AI 활용을 위한 추가 센서:

센서	용도	가격대	AI 활용
TDS 센서	전기전도도 측정	10,000원	영양소 농도 추정
pH 센서	산성도 측정	20,000원	수질 예측 필수
광센서 (BH1750)	조도 측정	3,000원	조명 최적화
암모니아 센서	암모니아 농도	30,000원	물고기 건강 예측
マイ크 모듈	펌프 소음 분석	2,000원	고장 예측

7. 개발 도구 및 프레임워크

7.1 추천 개발 스택

Edge AI 개발 스택

개발 환경

- Platform IO / Arduino IDE
- Python (모델 학습용)

ML 프레임워크

- Edge Impulse (추천 - GUI 기반, 초보자 친화적)
- TensorFlow Lite for Microcontrollers
- EloquentML (Arduino 친화적)

데이터 관리

- InfluxDB (시계열 데이터 저장)
- Grafana (시각화)

클라우드 연동 (선택)

- ThingSpeak
- AWS IoT Greengrass

7.2 Edge Impulse 활용 워크플로우

1. 데이터 수집 (ESP32에서)

↓

2. Edge Impulse Studio에 업로드

↓

3. 모델 설계 및 학습 (웹 기반 GUI)

↓

4. Arduino 라이브러리 다운로드

↓

5. ESP32에 배포

↓

6. 실시간 추론 실행

8. 참고 자료

8.1 튜토리얼 및 가이드

- [Edge Impulse 공식 문서](#)
- [TensorFlow Lite for Microcontrollers](#)
- [EloquentML Arduino 가이드](#)

8.2 관련 논문 및 연구

- "AI-Powered Sensors in Aquaponics" - MDPI Sensors Journal
- "Machine Learning for Fish Health Prediction" - Aquaculture Engineering
- "Time Series Forecasting for Water Quality" - Environmental Monitoring

8.3 커뮤니티

- [r/aquaponics](#) - 아쿠아포닉스 커뮤니티
 - [r/TinyML](#) - TinyML 커뮤니티
 - [Edge Impulse 포럼](#)
-



요약

if문 vs AI - 언제 무엇을 사용해야 하나?

상황	권장 접근법
단순 임계값 초과 경고 (수온 > 28°C → 경고)	if문 ✓
미래 수질 예측 (4시간 후 pH 예측)	AI 필수 ★
복합 조건 이상 탐지 (여러 센서의 비정상 조합 감지)	AI 필수 ★
물고기 건강 상태 분류 (건강/스트레스/위험)	AI 필수 ★
시스템 최적화 (에너지/생장/건강 균형 최적화)	AI 필수 ★
적응형 학습 (우리 시스템에 맞게 기준 조정)	AI 필수 ★

[!TIP] **시작하기 좋은 첫 번째 AI 프로젝트:** 이상 탐지 (Anomaly Detection) - 라벨링된 데이터 필요 없음 (비지도 학습) - Edge Impulse로 쉽게 구현 가능 - 즉각적인 실용적 가치 제공

작성일: 2026-01-15

버전: 1.0

관련 문서: [README.md](#), [disadvantages](#)