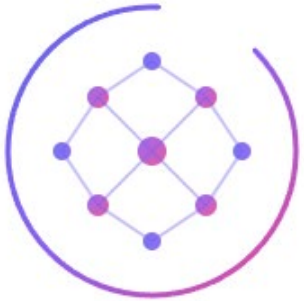


Open LLM with CPU

Open Source Language Models

DUTBOX

소개

과정 및 강사

목차

CPU LLM

LLM 목차

강사

강사

강사

이름: 최국현

메일: tang@linux.com, 회신은 다른 메일 주소로 드리고 있습니다. :)

사이트: tang.dustbox.kr

언제든지 질문 및 요청 환영 입니다.

LAB

랩 및 교육대상

LAB

현재는 오픈스택 기반으로 사용이 가능 합니다.

AI 소개

앤서블 기능

AI 소개-비용

최근 인공지능(AI)은 대규모 연산 자원과 고가의 GPU를 기반으로 발전해 왔습니다. 그러나 **AI의 본질은 알고리즘의 개방성과 효율성**에 있으며, 반드시 GPU만이 답은 아닙니다.

오픈소스 AI(Open Source AI) 는 이러한 문제의식을 바탕으로, 누구나 접근 가능한 **개방형 모델과 경량화된 실행 구조**를 통해 인공지능 모델 구성 및 운영에 대해서 낮은 비용 발생을 목표로 하고 있습니다.

자세한 내용은 아래 논문을 통해서 참고 부탁 드립니다.

https://www.arxiv.org/pdf/2509.18886

AI 소개-오픈소스 엔진

기존에는 GPU만이 대규모 AI 모델을 실행할 수 있다고 여겨졌습니다.

최근에는 **CPU 기반 최적화 기술(OpenVINO, llama.cpp, GGUF, quantization 기법 등)** 을 통해 **비용 효율적이고 유지보수가 용이한 AI 환경**을 구축하는 사례가 늘고 있습니다.

현재 대다수 기업들은 FPGA/NPU 혹은 ML전용 CPU를 구성하여 낮은 비용으로 기능을 구현을 꾀하고 있습니다.

- **장점**: 저비용, 전력 효율, 유지보수 용이성, 서버 자원 재활용 가능
- **활용 기술**: quantized 모델(4bit/8bit), SIMD 명령어 최적화, NUMA-aware scheduling
- **대표 사례**: Intel OpenVINO, llama.cpp, MLX, RWKV-Runner 등

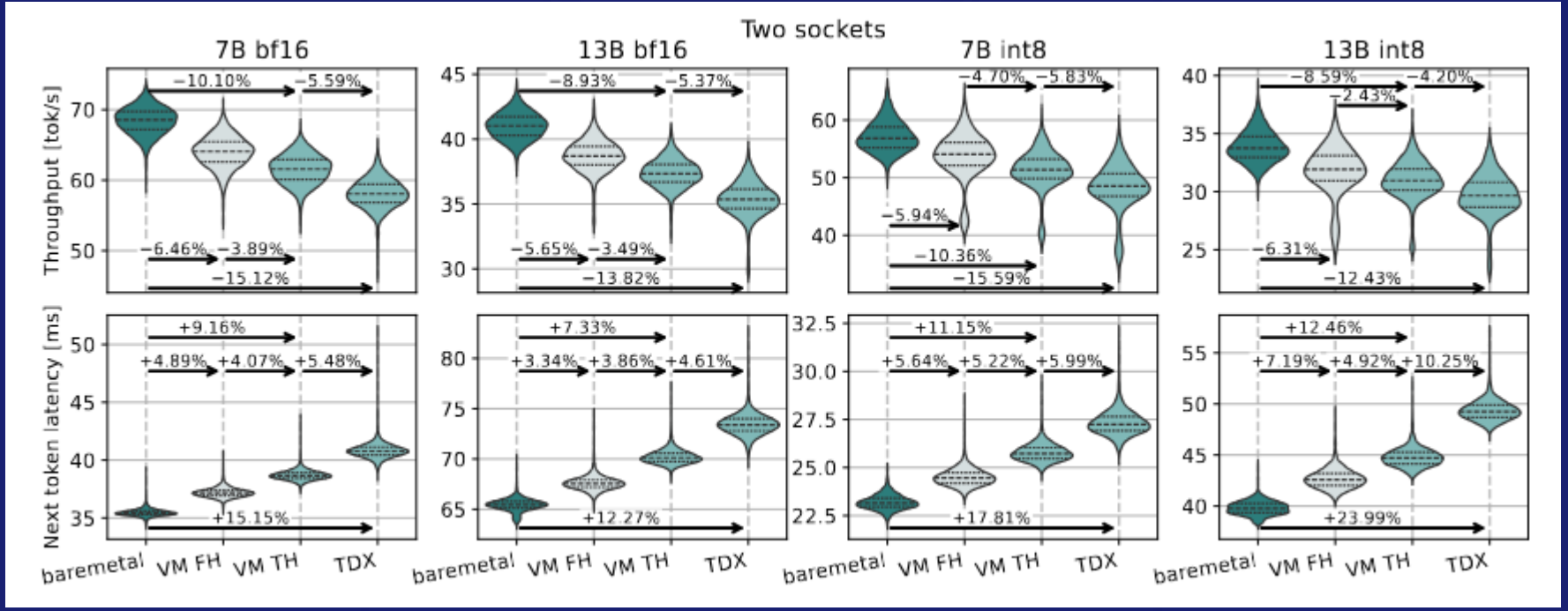
AI 소개 - GPU/CPU

GPU는 여전히 대규모 병렬 연산과 고속 학습(Training)에 있어 핵심적인 역할을 담당합니다. 반면 CPU는 **추론(Inference)** 중심 환경에서 효율적인 대안으로 부상하고 있습니다.

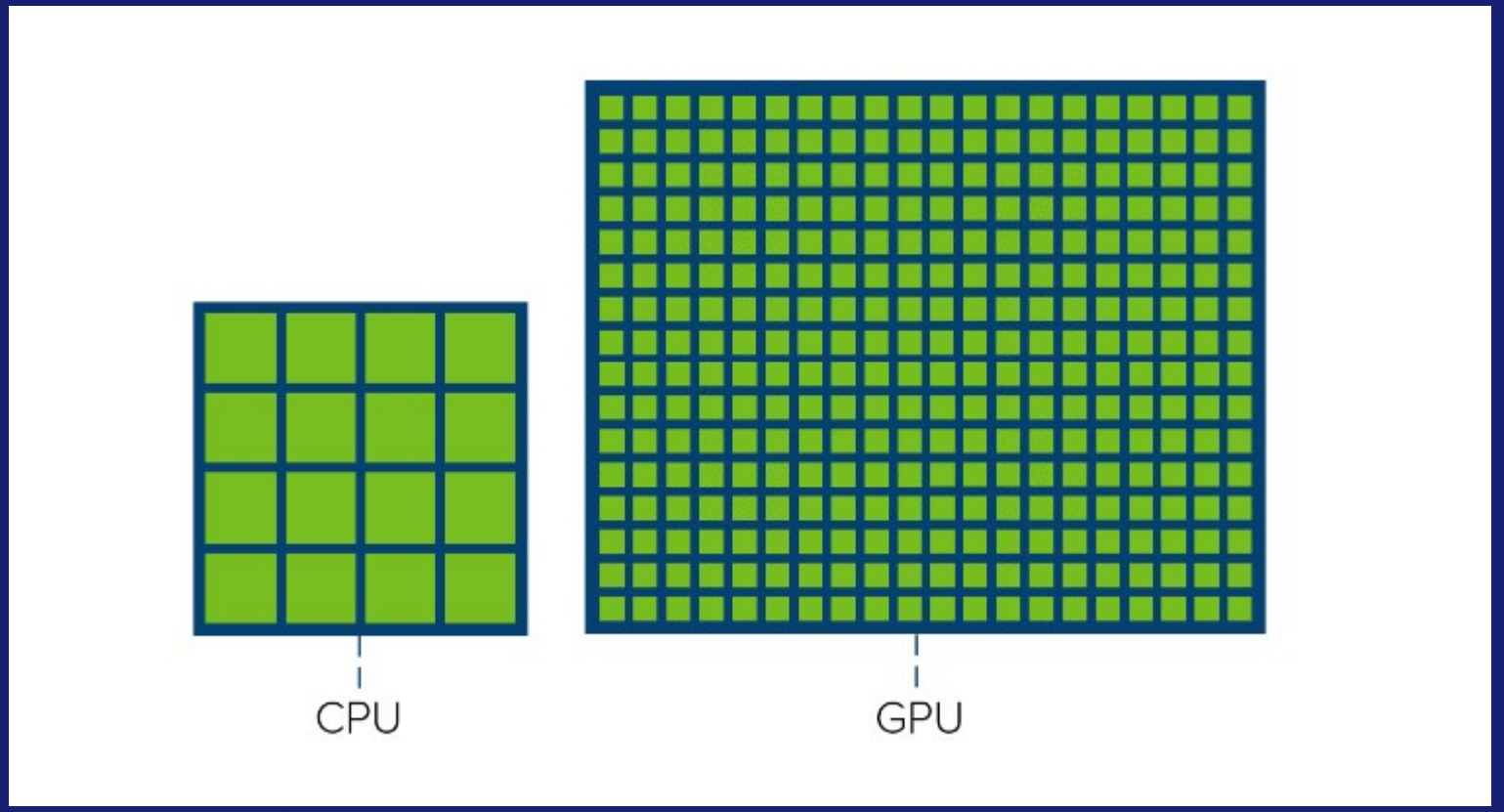
구분	GPU 기반	CPU 기반
주요 용도	대규모 학습(Training), 딥러닝 모델 훈련	경량 추론(Inference), 모델 배포
특징	높은 연산 병렬성, 고비용	범용성, 저비용, 유지보수 용이
장비 요구	고성능 GPU, 냉각/전력 인프라 필요	일반 서버 또는 VM 환경에서도 운영 가능
대표 기술	CUDA, cuDNN, TensorRT	OpenVINO, llama.cpp, oneDNN

https://arxiv.org/pdf/2309.02521

BAREMETAL/VM/GPU



CPU GPU INFERANCE



AI 오픈소스 도구

오픈소스 AI는 **폐쇄된 상용 모델(GPT, Claude 등)** 과 달리, 모델 구조·가중치·추론 코드가 공개되어 있으며, 커뮤니티 중심의 지속적 개선이 가능합니다.

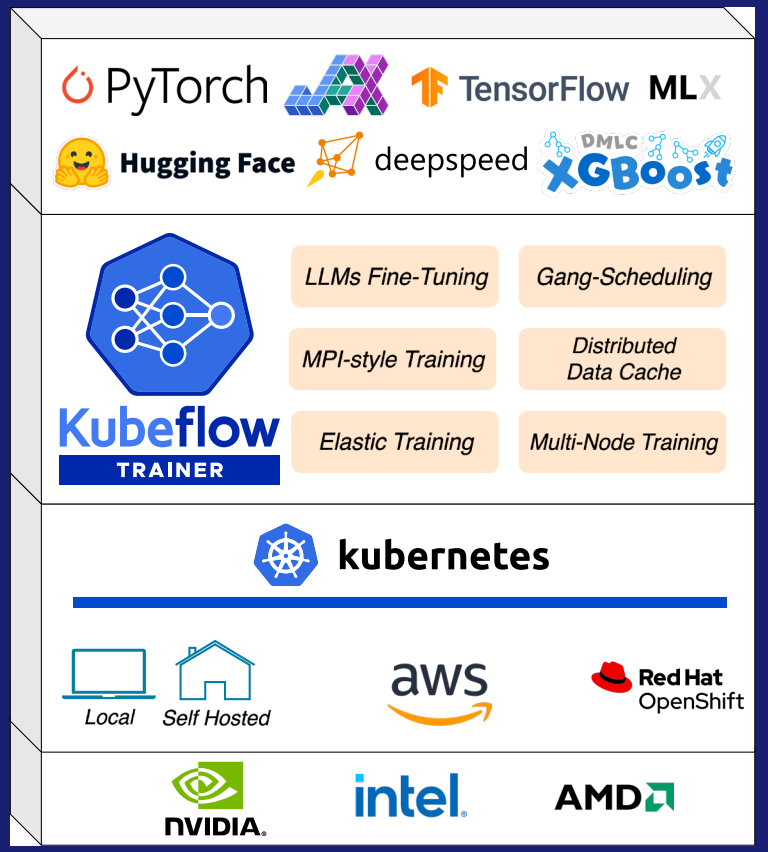
대다수 AI/ML 프로젝트는 오픈소스에서 시작이 되었으며, 이 프로젝트는 기존에 사용하던 APACHE 프로젝트 기반으로 많이 구성이 되어 있습니다.

대다수 AI/ML 프로젝트는 쿠버네티스 기반으로 사용하나, 기존의 가상머신 환경인 OpenStack/VMware vSphere와 같은 환경에서도 사용이 가능 합니다.

"AI의 진보는 데이터와 코드의 개방에서 시작된다."

AI 오픈소스 도구

관련기사:
https://www.cncf.io/announcements/2025/11/11/cncf-launches-certified-kubernetes-ai-conformance-program-to-standardize-ai-workloads-on-kubernetes/



AI/ML 오픈소스 도구

현재 다음과 같은 도구를 통해서 빠르게 학습 및 AI/ML 시스템 구축이 가능하다.

기능	FuseML	Kubeflow	MLFlow	Seldon
설치 난이도	매우 쉬움	어려움	쉬움	중간
전체 ML 파이프라인 자동화	Yes	Yes	No	No
모델 배포	Yes	Yes	No	Yes
GitOps	지원	제한적	No	No
커스터마이징	높음	낮음	낮음	중간
워크스페이스 UI	제한적	풍부	있음	없음

AI 구현 및 학습 도구

구분	주요 역할	기본 리소스	GPU 활용 여부
Kubeflow Control Plane	파이프라인 관리, 튜닝, 메타데이터 관리	CPU	GPU 불필요
FuseML	Kubeflow와 동일. 여기에 YAML 기반으로 통합적인 운영이 가능	CPU/GPU	선택 가능
데이터 전처리	CSV, 이미지 등 정규화·클리닝	CPU	가끔 가능하지만 드묾
모델 학습(Training)	신경망 모델 학습	GPU (기본)/CPU (가능)	GPU 중심
모델 검증 및 배포(Serving)	Inference, KFServing	CPU (기본)/GPU (선택적)	옵션

구축 시나리오

오픈스택

프로젝트
[LLM-PRJ]

MAGNUM KUBERNETES
CLUSTER

CONTROL
PLANE

TRAIN
PLANE

TRAIN INSTANCE
(CPU/GPU)

vGPU/MIG PARTITION

컴퓨트 호스트

vGPU/MIG Profile

GPU Server

ML FLOW
kubeFlow/FuseML

ML TRAINING

ML FLOW

Recrosses
CPU

MODEL
Training
[GPU]

Fine-Tuning

AI 엔진비교

항목	Llama (Meta AI)	OpenVINO Toolkit (Intel)
주요 기능	대형 언어 모델(LLM) → 텍스트 생성, 챗, 코드 생성 등 자연어 처리 중심	인공지능 모델의 **추론(inference) 최적화 및 배포** 툴킷 → 특히 CPU/엣지, 인텔 하드웨어에 최적화
사용처	텍스트 기반 언어모델 활용 → 챗봇, 코드생성, 자연어 인터랙션 등	학습된 모델(언어모델 포함 가능)을 다양한 하드웨어에서 빠르고 효율적으로 돌리기 위해 변환·최적화
오픈소스 여부/라이선스	**"오픈"**이라 불리지만 실제로는 오픈소스 라이선스 기준을 완전히 충족하지 않는다는 평가 메타의 홈페이지에 따르면 비독점, 전세계, 양도불가, 로열티 프리 형태 라이선스 제공됨 (AI Meta)	공식적으로 오픈소스 툴킷으로 제공됨. GitHub 리포지토리에 보면 “licensed under Apache License Version 2.0” 라고 명시되어 있음. (GitHub)

AI 엔진비교

항목	Llama (Meta AI)	OpenVINO Toolkit (Intel)
제약사항/특징	– 라이선스상 대형 서비스(예: 사용자 수가 많다거나 특정 사용처)에는 별도로 허가 필요하거나 제한이 있다는 조항 존재 (DEV Community) – 훈련 데이터 세트 등에 대한 공개가 충분치 않고 "오픈소스" 정의 기준(예: Open Source Initiative의 Open Source Definition) 에 부합하지 않는다는 비판 존재 (Open Source Initiative)	– 모델 추론 및 배포에 초점을 맞춤 → 예컨대 언어모델도 이 툴킷을 통해 배포 가능함 (OpenVINO Documentation) – Apache 2.0 라이선스 덕분에 비교적 자유롭게 수정·배포 가능
커뮤니티 및 활용도	언어모델 커뮤니티에서 활발히 활용됨. 다만 "완전한 오픈"이라는 표현에 대해서는 논란 있음 (미시간 온라인)	인텔 하드웨어 및 다양한 **엣지/클라우드 환경**에서 많이 쓰이고 있으며 문서·튜토리얼도 풍부함 (Intel)
적합한 사용 시나리오	언어모델을 직접 활용하거나 fine-tune 가능성이 있는 경우 다만 라이선스 조건을 잘 확인해야 함	이미 학습된 모델을 효율적으로 배포하고 최적화 하여 실서비스(특히 하드웨어 리소스가 제약된 환경)에서 활용하고자 할 때 적합

결론

Agent AI는 라이선스에 매우 민감하다. 직접 AI Engine를 제작하는 부분이 아니면, 아니면 라이선스에 대해서 많은 고민이 필요합니다.

해당 과정을 통해서 활용 및 프로세싱 과정을 확인하고, 어떠한 오픈소스 도구가 전략적으로 효율적인지 확인 합니다.

LLM 설치

llama.cpp

OpenVINO

안내

아래 내용은 설명 및 예시 입니다. 설치는 실제로 진행하지 않습니다.

수동 설치

설치는 빠르게 진행하기 위해서 파이썬 PIP를 통해서 llama.cpp를 설치한다. 이 방법은 테스트 용도로 적당하다.

제품 및 실무에서는 아래 방식으로 배포는 권장하지 않는다.

```
# dnf install -y python3.11
# python3.11 -m venv .venv
# source .venv/bin/activate

# pip install -U pip
# pip install -r requirements.txt
# pip install "llama-cpp-python[server]"
# python -m llama_cpp.server --model /path/to/model.gguf --n_ctx 4096
## 서버: http://127.0.0.1:8000/v1
```

컨테이너 기반 설치

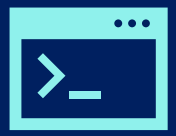
llama.cpp를 컨테이너 기반으로 사용하기 위해서 아래와 같이 수행한다. llama.cpp는 공식 이미지를 사용한다.

```
# buildah bud -t localhost/ai-infra-agent:latest \
-f Containers/Containerfile.agent
# podman play kube Containers/pod-kiki-ai-infra-agent.yaml --network podman
# podman pod ls
```

랩 구성

학습모델은 FuseML/KubeFlow를 지원합니다.
현재 **랩 및 교육 과정**에는 포함이 안되어 있습니다.

현재 모델은 llama.cpp로 구성이 되어 있습니다.
OpenVINO 추가 예정 입니다.



LLM 에이전트는 현재 **앤서블/오픈스택/쿠버네티스**를 지원하고 있습니다.
모든 코드는 YAML IaC 기반으로 지원하고 있습니다.
추후 Go 언어로 변경 예정 입니다.

활용코드

앤서블

오픈스택

에이전트 기능

시스템 운영을 위해서 LLM 에이전트가 필요하다. LLM 에이전트는 다음과 같은 역할을 지원한다.

1. 앤서블 플레이북 생성 및 실행
2. **오픈스택/쿠버네티스** 명령어 실행
3. 플레이북 혹은 **오픈스택/쿠버네티스**에 직접적으로 API 호출 및 실행
4. 자연어를 통한 YAML코드 생성

에이전트를 사용하기 위해서는 최소 한 개 이상의 AI Engine 구성이 필요하다. 여기서는 KIKI-AI-INFRA-AGENT를 통해서 해당 부분을 구성 및 구현하고 있다.

동작환경

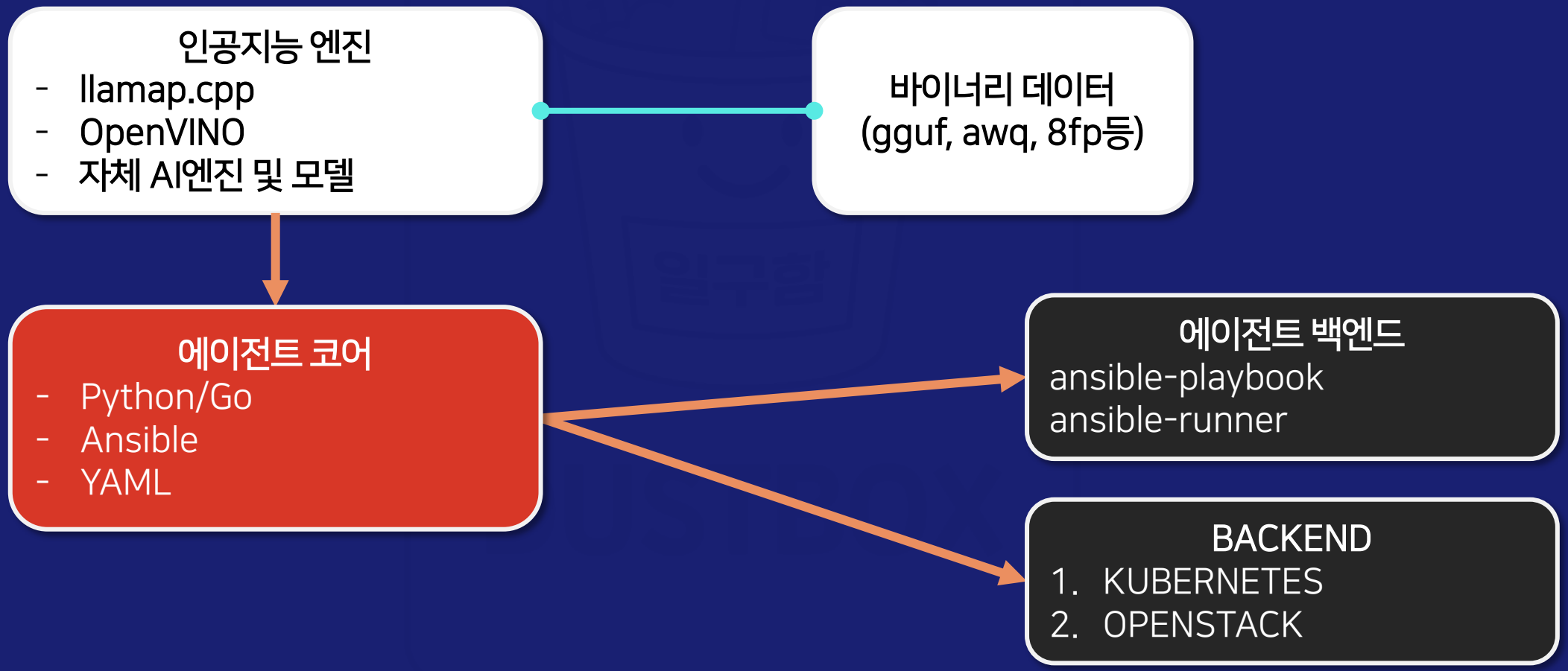
현재 **에이전트는** CPU 기반으로 동작. 동작하는 에이전트 인스턴스의 사양은 다음과 같이 구성이 되어있음.

- **CPU:** 8 vcpu
- **MEM:** 32 GiB
- DISK: 20GiB+

에이전트만 동작 시, 컴퓨팅 자원을 많이 소모하지 않지만, 랩에서는 **AI ENGINE + AGENT**가 통합된 상태로 동작하기 때문에 CPU 및 MEM가 높게 요구가 된다.

CPU 기반으로 사용하는 경우에는 Hyperthreading는 크게 도움되지 않는다. 물리적 코어 위주로 구성 후 사용을 권장한다. 빠르고 고성능을 원하는 경우 16개 코어를 권장한다.

KIKI-AI-AGENT 구조



LLM 에이전트 엔진

현재 사용하는 에이전트 AI는 두 가지를 지원하고 있다.

1. llama.cpp
2. OpenVINO

모델(model) 데이터는 Huggingface 혹은 OpenVINO에서 다운로드 가능 합니다. GPU가 없는 경우 CPU로 진행하여도 문제 없습니다.

설치 및 실행(컨테이너)

컨테이너 기반으로 실행 시, 다음과 같이 진행한다. 먼저, 이미지 빌드를 실행한다.

```
dnf install -y container-tools git
git clone https://github.com/tangt64/KIKI-AI-Infra-Agent
cd KIKI-AI-Infra-Agent
echo "ansible" > Containers/root_passwd.txt
buildah bud -t localhost/kiki-llm/kiki-ai-infra-agent:latest \
-f Containers/Containerfile.agent
buildah bud -t localhost/kiki-llm/kiki-web:latest \
-f Containers/Containerfile.web
```

설치 및 실행(컨테이너)

컨테이너 기반으로 실행 시, 다음과 같이 진행한다. 먼저, 이미지 빌드를 실행한다.

```
mkdir ~/models
curl http://10.0.0.250/granite-4.0-1b-Q4_K_M.gguf -o ~/models/granite-4.0-1b-Q4_K_M.gguf
setenforce 0
curl -L http://10.0.0.250/granite-4.0-1b-Q4_K_M.gguf.tar \
  | podman volume import kiki-models-pvc -
podman kube play Containers/pod-kiki_ai_infra_agent.yaml --network podman --replace
```

검증 명령어 및 클라이언트 설치

명령어는 다음과 같이 설치 및 확인한다.

```
# podman pod ls
# podman pod logs -f kiki-ai-infra-agent
# curl -s http://127.0.0.1:8082/v1/chat/completions -H "Content-Type:
application/json" \
-d '{"model":"local-llama","messages":[{"role":"user","content":"say
ok"}]}'
# mkdir -p ~/bin/
# cp kiki ~/bin/kiki
# chmod u+x ~/bin/kiki
# kiki
```

위치 옵션 설명

다음과 같이 옵션 사용이 가능 합니다.

```
kiki ansible-ai
kiki ansible-k8s
kiki ansible-osp
kiki chat "안녕?"
```

앤서블-명령어 사용 예제

앤서블 플레이북 생성은 다음과 같이 가능 합니다. 외부 인벤토리 파일도 사용이 가능 합니다.

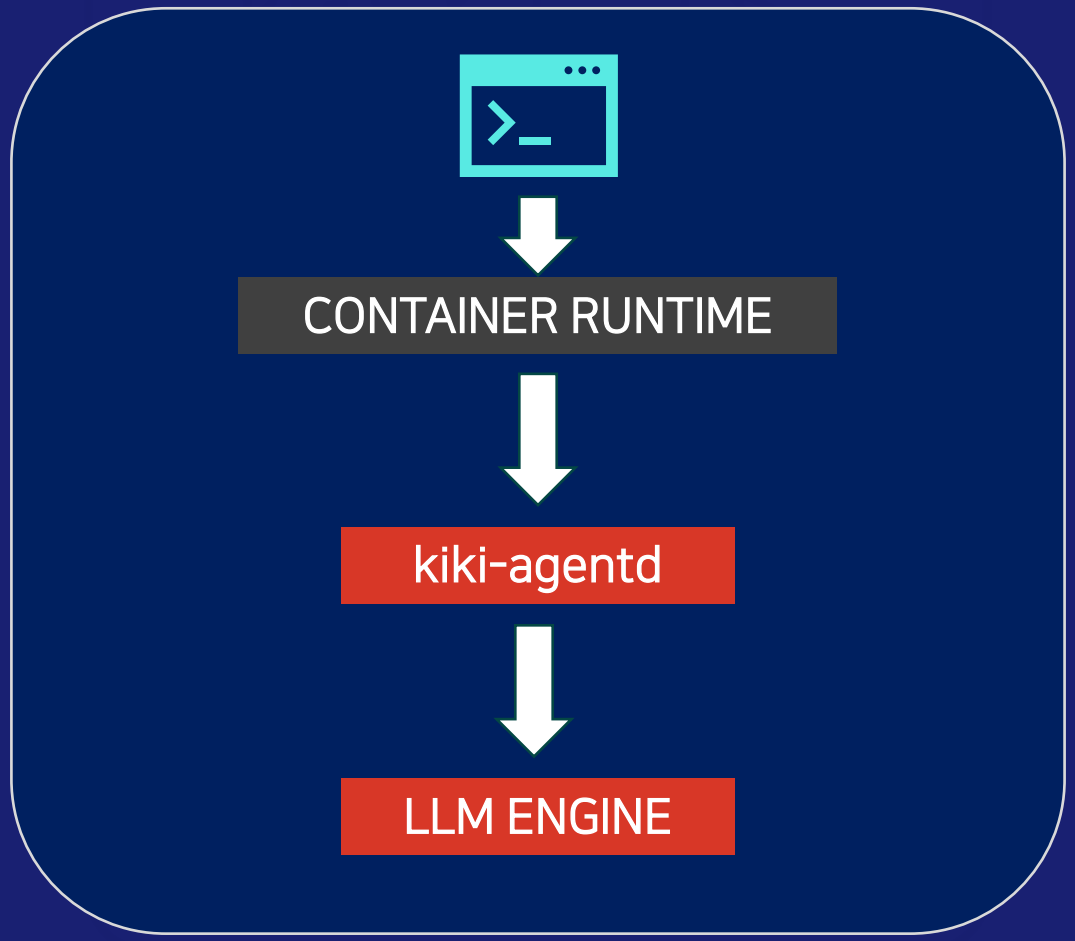
```
./kiki ansible-ai "httpd 설치하고 index.html 배포" \
  --target ansible \
  --inventory "node1,node2" \
  --verify syntax \
  --out playbooks/test-httpd.yml \
  --confirm \
  --apply
```

위치 옵션 설명

백엔드는 다음과 같이 지원하고 있다.

타겟	설명
chat	LLM 일반 대화
ansible-ai	자연어 → Ansible / OpenStack / Kubernetes / Heat YAML 생성
ansible-k8s	자연어 → Kubernetes용 Ansible 플레이북 (alias: ansible-ai --target k8s)
ansible-osp	자연어 → OpenStack용 Ansible 플레이북 (alias: ansible-ai --target osp
gen-role	Ansible role 스캐폴딩 생성 (roles/<name>/ 구조)
gen-k8s	Kubernetes Deployment + Service YAML 스캐폴딩 생성
gen-heat	OpenStack Heat 템플릿 스켈레톤 YAML 생성
health-collect	서버 헬스/로그 메트릭을 수집하여 SQLite DB에 저장
health-ai	수집된 메트릭을 기반으로 LLM이 상태를 요약/분석
log-ai	외부 txt 로그 파일 또는 stdin 로그를 LLM으로 분석

실행 구조



위치 옵션 설명

각 모든 옵션에 대한 상세 옵션은 `--help` 명령어로 확인이 가능 합니다.

다음단계!

KIKI 프로젝트의 다음 단계는 다음과 같이 준비 하고 있습니다.

1. **Go 리팩토링** 시 동일 구조 유지(Strategy 패턴: ansible vs direct)
2. **client-go/gophercloud**로 Direct 네이티브화 가능 (초기에는 기존 쉘 어댑터 재사용)
3. 공통 요약 JSON 유지 → 상위 로직/리포팅 변경 불필요
4. 단일 바이너리/동시성/배포 이점 확보