김도형 — 면접 발표 스크립트 & 예상 Q&A

면접 발표 스크립트 — 김도형 (15분)

> 대상: RL 기반 드론 제어 + 저해상도/고속 이동 인지 + Jetson 엣지 배포 포지션
> 구성: 자기소개·커리어 아크(2.5분) → 프로젝트 3개(각 3.5-4분) → 마무리(1.5분)
> 프로젝트 3개는 이 포지션의 3대 축에 1:1로 매핑되도록 선정:
> ① RL·sim-to-real (Quadruped) ② 저해상도 인지 (Depth SR, CVPR 1위) ③ 엣지 배포 (RT-DETR on Orin)
> 표기: 【슬라이드】 = 화면 큐 · 〈말〉 = 실제 발화 · ▶준비 = 면접관이 파고들 지점 대비

---

0. 오프닝 & 커리어 아크 — 2분 30초

【슬라이드 1】 이름 + 한 줄 태그라인: *"연구실의 AI를 실제 물리 시스템·엣지 디바이스로 옮기는 엔지니어"* / 로고 타임라인 (CERN → SNU → Neubility → RGA → OnLive Plus)

〈말〉
"안녕하세요, 김도형입니다. 오늘은 제 이력서를 그대로 읽는 대신, 제가 실제로 무엇을 만들었고 그게 이 포지션과 어떻게 연결되는지를 세 개의 프로젝트로 보여드리려 합니다.

제 커리어를 관통하는 한 문장은 이것입니다 — **'연구실 안에 머무는 AI를, 자원이 제한된 실제 하드웨어 위에서 실시간으로 돌아가게 만든다.'**

시작은 조금 특이합니다. 학부에서 **전기전자공학**을 전공했는데, 그중 제어공학 과목을 특히 잘했고 그 인연으로 **CERN에서 입자 가속기 제어와 프로그램 개발**을 했습니다. 여기서 '물리 시스템을 코드로 제어한다'는 감각을 처음 익혔습니다.

그러다 알파고와 이세돌의 대국을 보고 AI로 방향을 틀어, **서울대 AI 석박사 통합과정**에 들어갔습니다. 제 연구 주제는 처음부터 일관됐습니다 — **모델 경량화(model compression)**, 즉 '어떻게 하면 무거운 딥러닝을 작은 디바이스에서 돌릴까'였습니다. Spiking Neural Network 기반 객체검출, 양자화, 지식 증류가 그 시절 키워드였습니다.

이후 **Neubility에서 자율주행 배송로봇의 온디바이스 인지(AI Perception Engineer)**를 했고, **RGA에서는 CTO**로 컴퓨터비전·로보틱스 팀을 이끌었습니다. RGA에서는 모델만 만든 게 아니라 **사족로봇 센서 사양 선정, Go1 로봇 BMS, 부품 조달, JIRA 기반 프로젝트 관리, 그리고 7번의 VC 미팅과 특허 7건**까지 — 로봇 시스템을 처음부터 끝까지 세우는 일을 총괄했습니다. (현재는 OnLive Plus에서 웹·앱·백엔드 풀스택 제품을 총괄하고 있습니다.)

제가 이 오퍼를 수락한 이유부터 말씀드리겠습니다. 지금 웹·앱 제품을 총괄하고 있지만, 이게 결국 **화면 안의 서비스지 현실에 직접 닿는 게 아니라는 점**을 계속 고민하고 있었습니다. 요즘처럼 LLM이 잘하는 영역과 저를 차별화하고, 저 자신도 계속 발전할 수 있는 길은 결국 **현실 세계에 적용되는 AI**라고 생각했고, 그래서 로봇 쪽을 다시 들여다보던 참이었습니다. 그때 마침 이 오퍼가 왔고, 제가 해온 로봇·비전·엣지 경험이 이 회사에 실제로 기여할 부분이 있겠다고 판단해서 수락했습니다.

CTO를 하다가 왜 다시 엔지니어 포지션이냐 — 저는 오히려 전체를 총괄하면서 **여러 분야를 두루 이해하게 됐고, 그게 이 포지션에서 더 큰 도움이 된다**고 생각합니다. 그리고 솔직히 저는 포지션을 따지기보다, 늘 **제 앞에 놓인 문제를 해결하다 보니** 지금 여기까지 온 사람입니다. 저에게 중요한 건 과제가 크든 작든 그걸 제대로 풀어내고 **결과로 보여주는 것**이지 직함이 아닙니다. 열심히 하다 보면 인정을 받고, 포지션은 그 인정의 결과로 따라오는 것이라 생각하기 때문에 우선순위에 두지 않습니다.

그럼 제가 실제로 무엇을 풀어왔는지, 세 가지로 보여드리겠습니다."

▶준비: "CTO였는데 왜 엔지니어?" → 위 답변으로 선제 차단. 핵심 톤 = '총괄 경험으로 시야 넓어짐 + 포지션보다 문제 해결·결과가 우선, 포지션은 인정의 결과일 뿐'. 방어적이지 않게, 당당하게.
▶준비: "왜 우리 오퍼를 수락?" → 웹·앱은 화면 속 서비스라 현실에 안 닿음 / LLM과의 차별화 + 나의 성장 = 현실 적용 AI(로

봇) / 기여할 수 있다고 판단. (돈·조건 언급 금지, 미션 정렬로.)
▶준비: "최근 RL을 얼마나 직접 했나?" → 프로젝트 1에서 '제품 수준으로 오래 운용했다'가 아니라 '설계·비교·구현 수준으로 했다'고 정직하게 프레이밍(다음 섹션 참고). CV에도 transformer를 RL '대안'으로 탐색했다고 적혀 있으니 이 톤이 CV와 일치.
▶준비: 센서 선정 경험(사족로봇 카메라·IMU·SoC 사양 검토)은 드론 카메라/IMU 질문("왜 이 카메라·global vs rolling shutter·IMU 주파수")과 직결 — 이 한 줄을 반드시 심어두기.

---

1. 프로젝트 ① — 영상에서 로봇으로: RL 기반 사족보행 (RAC → Quadruped) — 4분

왜 이 프로젝트를 먼저 보여드리나: 이 포지션의 핵심인 **RL + 모션 imitation + sim-to-real**이 한 프로젝트에 다 들어있기 때문.

【슬라이드 2】파이프라인 다이어그램: `동물 영상 크롤링 → 관절 추출 + RAC 3D 복원 → IK 리타게팅 → MPC gait 선택 + 모션 모방(시뮬) → 도메인 적응 → 실기체` / 우측 작은 박스: `transformer POC = RL 대안 탐색` / 실제 사족로봇 gait GIF

〈말〉
"첫 번째는 RGA에서 한 **영상 기반 민첩 사족보행(agile locomotion by imitating animals)** 프로젝트입니다. 문제의식은 이거였습니다 — 로봇에게 자연스러운 보행을 가르치려면 보통 모션캡처 장비가 필요한데, 이게 비싸고 세팅이 까다롭습니다. 그래서 저희는 **유튜브에 있는 평범한 동물 영상만으로 로봇의 보행 데이터를 만들고 정책을 학습**시키는 파이프라인을 만들었습니다.

흐름은 이렇습니다.
첫째, 동물 영상을 크롤링해 딥러닝으로 **관절 동역학·움직임 특징을 추출**하고, **RAC(Reconstructing Animatable Categories)**로 단안 영상에서 3D 골격과 프레임별 관절 움직임까지 복원합니다 — 개체별 형태 차이를 담는 morphology code, articulation, soft deformation이 들어갑니다.
둘째, 추출한 관절 궤적을 **역기구학(IK)**으로 로봇 관절에 리타게팅해 현실적인 데이터셋을 만듭니다.
셋째, **MPC로 gait를 선택**하고, 시뮬레이션(PyBullet/Mujoco)에서 모션 모방을 학습합니다. 보상은 pose·velocity·end-effector·root 궤적을 각각 지수 커널로 매칭하는 형태로 설계했습니다.
넷째, **도메인 적응** — 마찰·질량·모터 파라미터를 randomize하고, 환경 동역학을 담는 latent embedding을 실기체에서 미세조정해 sim-to-real gap을 줄입니다.

여기에 더해, **transformer 기반 사족보행 POC**를 따로 만들었습니다. 목적은 'RL이 sample efficiency가 낮고 시뮬 비용이 크니, 이미 있는 보행 궤적을 sequence로 모방하는 transformer가 RL의 대안이 될 수 있는가'를 검증하는 것이었고, vanilla transformer로 초기 실험까지 돌렸습니다.

RAC 복원 품질 자체도 baseline(BANMo, ICON 등) 대비 Chamfer Distance를 9점대에서 6.0으로, F-score를 크게 끌어올렸습니다.

이 프로젝트에서 제가 배운 핵심은 **정책 자체보다 그 주변이 성패를 가른다**는 겁니다. 데이터 파이프라인, 보상 설계, IK 리타게팅의 좌표계·스케일, gait 선택, 도메인 randomization 항목 선정 — 이게 실제로 로봇을 걷게 만들었습니다."

▶준비 (면접관이 팔 지점):

- "PPO/SAC/TD3 중 뭘 썼나, reward는 어떻게 설계했나?" → 시뮬 병렬 학습·안정 baseline엔 PPO, 실기체 데이터가 비싸지면 sample efficiency 위해 SAC 검토. reward는 위 5개 term(pose·vel·end-effector·root pose·root vel) 가중합. **직접 오래 운용이 아니라 imitation 기반 설계·구현·비교 + transformer/MPC 대안 탐색을 했다고 정직하게.**
  GAE·entropy·clipping 정의는 개념 수준으로 방어 가능하게 복습.
- "sim-to-real gap을 실제로 어떻게 좁혔나?" → domain randomization(마찰/질량/모터/센서노이즈/지연) + latent z 실기체 적응. '학습 잘됐으니 된다'는 답 금지.
- "왜 RL 대신 imitation/Transformer도 검토했나?" → RL은 sample efficiency 낮고 시뮬 비용 큼, 이미 존재하는 궤적을 imitation으로 학습하는 게 더 실용적일 수 있어 비교 검토.
- **드론으로의 연결 한마디 반드시 언급:** "이 sim-to-real·도메인 적응·safety 경계 설계 경험이 드론 제어 정책을 실기체로 옮기는 데 그대로 적용됩니다."

2. 프로젝트 ② — 저해상도·노이즈 깊이 복원 (CVPR 2024 AIS Challenge 1위) — 3분 30초

왜: 이 포지션의 **저해상도 인지** 문제를 정면으로 다룬 실적. 게다가 국제 챌린지 1위 크리덴셜.

【슬라이드 3】좌: 노이즈 낀 저해상도 depth / 우: 복원 결과 vs GT / 배지: **1st Place — CVPR 2024 AIS Depth Compression·Upsampling·Refinement Challenge** (+ *3rd Place, CVPR AIS Monocular Depth Estimation Challenge*) / 아키텍처: `[Depth Anything 상대깊이 + LR depth + RGB] → 이중경로 NAFNet 인코더 → AdaIN 융합 → U-Net 디코더`

〈말〉
"두 번째는 **CVPR 2024 AIS Depth 챌린지에서 1위**(Compression·Upsampling·Refinement)를 한 프로젝트입니다. 같은 워크숍의 Monocular Depth Estimation 챌린지에서는 3위도 했습니다. 이 포지션과 직접 닿는 이유는, 문제 자체가 **'저해상도에다 노이즈까지 낀 입력에서 어떻게 신뢰할 만한 정보를 복원하느냐'**이기 때문입니다. 드론이 흔들리는 화면에서 작은 객체를 봐야 하는 상황과 본질이 같습니다.

참고로 이 작업 전에 저는 **Depth Anything 논문을 밑바닥부터(from scratch) 직접 구현**하면서 monocular depth의 원리와 Transformer 인코더 구조를 손으로 익혔습니다. 그 위에서 '상대 깊이를 metric 깊이로 변환'하는 문제까지 다뤘기 때문에, 아래 파이프라인이 단순히 남의 모델을 갖다 쓴 게 아니라는 점을 먼저 말씀드립니다.

핵심 아이디어는 **정보가 없는 한 장을 억지로 확대하지 않는다**는 것이었습니다. 대신 두 가지 보조 신호를 붙였습니다.
하나는 **Depth Anything** foundation model에서 뽑은 **상대 깊이(relative depth)** — 절대값은 부정확해도 구조·경계 정보가 살아있습니다.
다른 하나는 원본 RGB. 이 셋(상대깊이·LR depth·RGB)을 concat해서 넣습니다.

아키텍처는 U-Net 골격에 **이중 경로 인코더**(상대깊이 경로·LR depth 경로 각각 NAFNet 블록)를 두고, 두 경로의 분포를 **AdaIN으로 정렬·융합**한 뒤 디코더에서 복원합니다. 손실은 픽셀 L1에 **Sobel 기반 edge preservation loss**를 더해 경계 디테일을 살렸습니다.

결과는 A6000 한 장으로 3일 학습, RTX 3090에서 **약 24 FPS, 29M 파라미터**로 준실시간이 나왔고, 노이즈가 심한 LR 입력에서도 baseline보다 확연히 GT에 가깝게 복원했습니다.

여기서 얻은 교훈이 드론 인지에 그대로 옵니다 — **저해상도 문제는 단일 프레임 초해상도로 풀리지 않고, foundation model의 사전지식·보조 신호·시간축 정보를 융합해야 한다**는 것입니다. 그리고 super-resolution 결과를 인식 근거로 쓸 땐 hallucination 위험이 있어서, 사람에게 보여주는 용도가 아니라 detector 입력 보조로 제한하고 원본 기반 confidence를 함께 관리해야 한다는 것도 이때 체득했습니다."

▶준비:

- "왜 Depth Anything인가? relative vs metric depth 차이?" → 상대깊이는 스케일 모호하지만 구조/경계 prior가 강함. 여기선 절대깊이 회귀가 아니라 SR 가이드로 사용.
- "AdaIN을 왜 융합에 썼나?" → 두 경로의 통계 분포가 달라서 정렬 필요. σ/μ 정규화로 상대깊이 특징을 LR 분포에 맞춤.
- "실시간성 근거?" → 29M/24FPS 수치 제시, 엣지로 가면 TensorRT·경량화 추가 필요(프로젝트 ③로 자연 연결).
- **드론 연결:** "드론의 저해상도·모션블러 인지도 같은 철학 — ROI에만 선택적 SR, foundation prior, 시간축 융합으로 접근하겠다."

---

3. 프로젝트 ③ — Jetson Orin 실시간 검출 배포 (RT-DETR 경량화) — 3분 30초

왜: 이 포지션이 명시한 **Jetson·distillation·quantization·TensorRT 엣지 배포**를 정확히 수행한 실적.

【슬라이드 4】성능표 (RT-DETR 46.5mAP/20M/60B → **Ours-v3 48.0/17M/23B**, Ours-v0 41.2/**5.2M/6.4B**) / 배포 파이프라인: `PyTorch → ONNX → TensorRT(FP16/INT8) → DeepStream on Orin`

〈말〉
"세 번째는 **Nvidia Orin 위에서 실시간으로 도는 객체검출기**를 만든 프로젝트입니다. 앞의 두 개가 '무엇을 인식하느냐'였다면, 이건 '제한된 하드웨어에서 어떻게 실제로 돌리느냐'입니다.

baseline은 **RT-DETR**을 골랐습니다. YOLO와 달리 **NMS 후처리가 없는 end-to-end** 구조라 파이프라인이 단순하고 지연이 예측 가능하다는 게 엣지에서 큰 장점이었습니다. 여기에 anchor-free 검출의 장점을 보려고 **FCOS head를 붙여 실험**하고, **backbone도 새로 교체**해가며 정확도·속도 트레이드오프를 탐색했습니다.

여기서 두 방향으로 경량화했습니다.
정확도를 올린 버전(**Ours-v3**)은 feature map 재사용으로 FLOPs를 줄이면서 오히려 mAP를 **48.0**으로, 원본 RT-DETR의 46.5를 넘겼습니다 — 파라미터는 20M에서 17M, FLOPs는 60B에서 23B로.
극단적으로 경량화한 버전(**Ours-v0**)은 **5.2M 파라미터·6.4B FLOPs**까지 내려서, Orin 같은 제약 환경에서 돌 수 있게 했습니다.

배포는 **PyTorch → ONNX → TensorRT** 순으로, FP16을 baseline으로 잡고 INT8은 그 다음에 검증하며 올렸습니다. 이건 의도적인 순서입니다 — 저해상도·작은 객체는 feature margin이 작아서, calibration 데이터가 부실하면 INT8에서 성능이 크게 무너지기 때문입니다. 최종적으로 **DeepStream 파이프라인**에 통합해 멀티카메라 실시간 스트리밍까지 붙였습니다.

Neubility 시절엔 같은 일을 Xavier에서 했는데, 거기선 모델뿐 아니라 **GStreamer 파이프라인 최적화, SW 디코딩을 HW 디코딩으로 교체해 CPU 오버헤드를 줄이는** 시스템 레벨 작업까지 했습니다. 그래서 저는 **모델 추론 시간만 재는 게 아니라 camera-to-command end-to-end latency를 본다**는 원칙을 갖고 있습니다."

▶준비:

- "왜 YOLO 아니고 RT-DETR?" → NMS 제거·end-to-end·예측 가능한 지연. 단점(연산량↑)도 인정하고 그래서 경량화.
- "PTQ에서 무슨 문제 있었나? FP16이 왜 느릴 때가 있나?" → calibration dataset 품질·activation saturation·accuracy 저하 / unsupported operator, CPU-GPU memory copy, CUDA sync. **경험 기반으로 답.**
- "FPS는 높은데 제어가 늦게 반응한다면?" → throughput≠latency. NMS·전처리·디코딩·큐 적체. latest-frame 방식으로 오래된 프레임 버림.
- "Jetson 하나로 detection+tracking+RL+PX4 다 돌리라면?" → (최종 통합 질문) 아래 마무리에서 그림으로 답.

---

4. 마무리 — 세 프로젝트가 이 포지션에 어떻게 합쳐지는가 — 1분 30초

【슬라이드 5】통합 시스템 다이어그램:
`Camera → (경량 Detector 10Hz) → (Tracker + IMU ego-motion 30Hz) → State Estimation → RL Policy 20Hz → Safety Layer(clip/geofence/fallback) → PX4 자세·rate 제어 → Motor`
아래 캡션: TensorRT · FP16 · ROI 재검출 · async pipeline · zero-copy · safety fallback

〈말〉
"정리하면, 오늘 보여드린 세 개는 사실 이 포지션 하나로 모입니다.

- 사족보행 프로젝트의 **RL·모션 모방·sim-to-real** 경험은 드론 제어 정책을 실기체로 안전하게 옮기는 부분에,
- Depth 챌린지의 **저해상도·노이즈 인지** 경험은 흔들리는 드론 화면에서 작은 객체를 보는 부분에,
- RT-DETR·Xavier의 **엣지 배포·경량화** 경험은 Jetson 하나로 다 돌려야 하는 제약에 직접 대응합니다.

만약 'Jetson 하나로 detection·tracking·RL·PX4를 다 돌리라'고 하신다면, 저는 이렇게 설계하겠습니다 — 무거운 검출기를 매 프레임 돌리지 않고 **경량 detector를 저주기(10Hz)로, tracker와 IMU 기반 ego-motion 보상을 고주기로** 돌립니다. 작은 객체는 원본 ROI만 잘라 재검출하고, RL 정책은 모터를 직접 때리지 않고 목표 속도·자세를 내보내 **저수준 제어는 검증된 PX4에** 맡깁니다. 그 사이엔 항상 **safety filter와 fallback controller**를 둬서, 정책이 학습 분포 밖으로 나가거나 추론이 타임아웃되면 안전하게 회복합니다.

저는 '저해상도 이미지를 경량 AI가 한 번에 정확히 인식'하는 구조는 성공 확률이 낮다고 봅니다. 대신 **카메라·IMU 동기화, ego-motion 보상, ROI 재검출, 시간축 융합, detector와 tracker 분리, 그리고 safety 경계**를 함께 설계해야 실사용에 닿는다고 생각하고, 그게 제가 이 팀에서 하고 싶은 일입니다. 감사합니다."

▶준비: 이 마지막 다이어그램이 사실상 최종 질문("어떻게 다 돌릴 거냐")의 선제 답변. 여기서 async pipeline·zero-copy·CUDA stream·latest-frame까지 한 마디씩 얹으면 강함.

부록 — 발표 운영 팁

- **총 5장 슬라이드**로 충분(오프닝 1 + 프로젝트 3 + 통합 1). 슬라이드는 수식 나열 금지, 다이어그램·표·GIF 중심.
- 시간 배분 여유: 프로젝트당 30초 버퍼. 늦어지면 프로젝트 ①(RL)은 사수, ②③는 결과 수치만 남기고 압축.
- **정직성 원칙:** RL은 "제품 수준 장기 운용"이 아니라 "설계·구현·비교 + transformer/MPC 대안 탐색" 수준으로 프레이밍 — CV 문구와도 일치하고, 면접관이 GAE/entropy/clipping까지 파고들 때 방어 가능한 선.
- 숫자는 반드시 기억: `RT-DETR Ours-v3 48.0 mAP / 17M / 23B`, `Ours-v0 5.2M / 6.4B`, `Depth 24FPS / 29M`, `RAC CD 6.0`, `SNN 오차 <5%`, `CVPR 2024 Depth 1위 + Monocular 3위`, `학부 상위 0.16% Summa Cum Laude`.
  - ⚠️ RT-DETR의 mAP/파라미터/FLOPs 수치는 **사이트(fpost) 기준**이며 CV 본문엔 없음 — 발표에서 인용은 OK지만, 면접관이 "논문/공식 수치냐"고 물으면 "우리 내부 실험 결과(Ours-v0~v3)"라고 명확히.
- **CV vs 사이트 정합성 (직접 결정 필요):** 제출한 `Dohyeong_CV_2025.pdf`의 최근 이력은 **RGA CTO(~2024.10)에서 끝남 — OnLive Plus/CashPocket 없음.** 반면 사이트엔 OnLive CTO가 대표 프로젝트로 있음. 면접관이 CV를 들고 오므로, OnLive를 "현재 진행 중"으로 한 줄만 언급하고 깊이 들어가지 말 것(드론 RL과도 무관). 원하면 CV에 OnLive 한 줄 추가해 정합 맞추는 것 권장.
- **SLAM/Tracking/Face 프로젝트**도 CV에 있음(LiDAR visual SLAM·OctoMap voxelization / object tracking / AM-RADIO 표정인식). 3개 딥다이브엔 안 넣었지만 면접관이 팔 수 있으니 통합 다이어그램(tracking)·Q&A로 대비. SLAM: voxelization·OctoMap vs occupancy grid·point cloud filtering / Tracking: ByteTrack·Kalman·Hungarian·ID switching.
- 영어 발표가 필요하면 이 스크립트 그대로 영어판으로 변환 가능(요청 시).
- 근거: 사이트 `fpost/*` + `Dohyeong_CV_2025.pdf`(외장 T7 드라이브) 대조 작성 완료. 수치·표현은 CV/사이트와 정합.

Part 2 — 예상 질문 & 답변 (기술 면접 대비 Q&A)

> 발표(Part 1) 후 이어질 기술 토론 대비. 답변은 **짧고 단단하게**, 모르면 "그 부분은 개념 수준으로 이해하고 있다"고 정직하게.
> ★ = 반드시 암기 (면접관이 10분 이상 파고들 가능성 높은 핵심 주제)

A. 배경 · 동기

Q. 간단히 자기소개 해주세요.

> AI를 실제 제품에 적용하는 것을 목표로 연구와 개발을 해왔습니다. 서울대학교 AI 석박사 과정에서 Computer Vision과 model efficiency를 연구했고, 이후 Neubility와 RGA에서 로봇 비전 — Object Detection, Tracking, Depth Estimation, Model Compression, Jetson Deployment — 를 수행했습니다. 또한 Transformer 기반 Robot Learning과 On-device AI에 집중했습니다.

Q. CTO였는데 왜 다시 Engineer로 오시나요? (톤: 방어적이지 않게, 당당하게)

> 전체를 총괄하면서 여러 분야에 대한 이해도가 높아졌고, 그게 이 포지션에서 오히려 더 도움이 된다고 생각합니다. 저는 포지션을 따지기보다 늘 제 앞에 놓인 문제를 해결하다 보니 여기까지 왔습니다. 과제가 크든 작든 제대로 풀어 결과로 보여주는 게 중요하지 직함이 중요하지 않습니다. 열심히 하다 보면 인정을 받고, 포지션은 그 인정의 결과로 따라오는 것이라 생각해 우선순위에 두지 않습니다.

Q. 왜 우리 오퍼를 수락했나요? (돈·조건 말고 미션 정렬로)

> 지금 웹·앱 제품을 총괄하고 있지만 결국 화면 속 서비스지 현실에 직접 닿는 게 아니라는 점을 고민하고 있었습니다. LLM이 잘하는 영역과 저를 차별화하고 저 자신도 계속 발전할 수 있는 길은 현실 세계에 적용되는 AI, 즉 로봇이라고 생각해 그쪽을 다시 들여다보던 참이었습니다. 그때 이 오퍼가 왔고, 제 로봇·비전·엣지 경험으로 기여할 부분이 있겠다고 판단해 수락했습니다.

B. Reinforcement Learning ★ (가장 많이 나올 영역)

Q. RL은 실제 어느 정도 경험하셨습니까? (정직 프레이밍)

> RL 알고리즘 자체를 연구한 경험도 있지만, 실제 프로젝트에서는 RL을 로봇 시스템에 적용하기 위한 구조 설계와 locomotion 연구를 주로 했습니다. 특히 MPC와 RL을 비교하면서 Transformer 기반 imitation learning 가능성도 검토했습니다.

Q. PPO 써보셨습니까?

> PPO를 장기간 제품에 직접 적용한 경험보다는, PPO의 구조와 장단점을 이해하고 있습니다. locomotion 프로젝트에서는 RL 기반 접근과 Transformer 기반 imitation learning을 비교하는 역할을 수행했습니다.

Q. PPO와 SAC 차이? ★

> **PPO** — On-policy, 안정적, 구현 단순, sample efficiency 낮음(같은 데이터 한 번만 사용).
> **SAC** — Off-policy, Replay Buffer로 데이터 재사용해 sample efficiency 높음, entropy term으로 exploration 우수.

Q. PPO / SAC / TD3 비교하고, 드론이면 무엇을 선택? ★

> PPO는 On-policy로 학습이 안정적이고 구현이 단순하지만 sample efficiency가 낮습니다. SAC는 Off-policy라 replay buffer로 데이터를 반복 사용해 sample efficiency가 높고 entropy term으로 exploration이 좋습니다. TD3도 Off-policy지만 deterministic policy를 쓰며 DDPG의 Q-value overestimation 문제를 해결한 알고리즘입니다.
> 드론처럼 실제 데이터 수집 비용이 큰 환경에서는 SAC가 유리하지만, **초기 baseline은 PPO가 안정적**입니다. 다만 실제 시스템에서는 RL 알고리즘 자체보다 **Safety Layer와 Controller 구조가 더 중요**하다고 생각합니다.

Q. 왜 PPO는 안정적입니까?

> 정책을 한 번에 크게 업데이트하지 않고 Clipping으로 정책 변화량을 제한하기 때문입니다.

Q. 왜 SAC가 sample efficiency가 높습니까?

> Replay Buffer로 같은 데이터를 여러 번 학습할 수 있기 때문입니다.

Q. 왜 Entropy를 넣습니까?

> 초기에 다양한 행동을 탐색하도록 유도해 local optimum에 빠지는 걸 줄이기 위해서입니다.

Q. 왜 RL이 직접 Motor를 제어하면 안 됩니까? ★ (거의 반드시 나옴)

> 세 가지 때문입니다. ① **안전성** — 예상치 못한 action을 낼 수 있음. ② **설명 가능성(Explainability)** — 왜 그 action을 냈는지 설명이 어려움. ③ **검증(Verification)** — Safety Certification이 어려움.
> 그래서 실제 드론은 `RL → Reference → MPC/PID → Motor` 처럼, RL은 상위 정책, 저수준 제어는 검증된 컨트롤러가 담당하게 구성하는 게 일반적입니다.

C. Transformer 기반 Quadruped

Q. 왜 RL 대신 Transformer를 하셨습니까?

> RL은 학습 비용이 매우 크고 실제 locomotion 데이터를 활용하기 어렵습니다. 그래서 이미 존재하는 locomotion trajectory를 imitation learning으로 학습할 수 있는지 검토했습니다.

Q. 입력은? → Joint angle · Joint velocity · Base velocity · Previous action · Desired command
Q. 출력은? → Target joint 또는 Residual action
Q. Transformer 장점은? → 긴 sequence 학습 · Temporal dependency · Large dataset 활용

D. Video → Robot (Imitation Learning)

Q. 동물 영상으로 locomotion을 어떻게 학습했습니까?

> 동물 영상을 수집한 뒤 pose estimation을 수행하고, inverse kinematics로 joint trajectory를 복원한 다음 imitation learning으로 학습하는 방향을 연구했습니다.
> 흐름: `Video → Pose Estimation → Skeleton → Inverse Kinematics → Joint Trajectory → Imitation Learning`

Q. 왜 animal입니까? → 다양한 locomotion pattern을 확보할 수 있기 때문입니다.
Q. 왜 inverse kinematics가 필요합니까? → 영상에는 joint angle 정보가 직접 없기 때문입니다.

E. Object Detection (RT-DETR)

Q. 왜 RT-DETR을 선택했습니까?

> Transformer 기반 detector의 가능성을 검토하려고 선택했습니다. YOLO보다 계산량은 늘지만 NMS 제거와 높은 detection accuracy가 장점입니다.

Q. 왜 FCOS Head를 붙였습니까? → Anchor-free detector 구조를 비교하기 위한 POC였습니다.
Q. RT-DETR 단점은? → 계산량 · Memory · Jetson에서의 latency.

F. Tracking

Q. Tracking은 어떻게 구현했습니까?

> Detection과 Tracking을 분리했습니다. Detection으로 bounding box를 만들고, Tracking은 Kalman prediction과 association으로 ID를 유지합니다.
> 흐름: `Camera → Detector → Bounding Box → Kalman Prediction → Association → Tracking ID`

Q. Kalman Filter 역할? → Detection이 없는 프레임에서도 객체 위치를 예측합니다.
Q. Association은? → IoU + Hungarian Matching.
Q. ID Switch는 왜 발생? → Detection 오류, Occlusion, Association 오류 때문입니다.

G. Jetson / TensorRT / ONNX / Optimization ★

Q. Jetson에 모델을 어떻게 배포했습니까?

> `PyTorch → ONNX Export → TensorRT Engine → FP16 Benchmark → INT8 Calibration → Profiling → Jetson Deployment`

Q. TensorRT가 왜 빠릅니까? → Layer Fusion · Kernel Optimization · Memory Optimization · GPU/CUDA 최적화.
Q. FP16과 INT8 차이? → FP16은 정확도 손실이 거의 없고, INT8은 calibration이 필요하지만 더 빠름.
Q. INT8 Accuracy가 떨어지는 이유 / PTQ 문제? → Calibration dataset이 부적절하면 activation distribution이 깨져 정확도가 크게 떨어집니다.
Q. 왜 ONNX가 필요? → Framework 독립적으로 배포하기 위해서.
Q. 변환이 안 되는 경우? → Unsupported Operator · Dynamic Shape.

Q. Jetson에서 FPS가 안 나옵니다. 무엇부터? ★

> 순서대로 확인합니다 — ① GPU Usage ② Memory ③ CPU 병목 ④ TensorRT ⑤ Camera Decode ⑥ NMS ⑦ Memory Copy.

Q. Zero Copy가 왜 필요? → CPU↔GPU memory copy를 줄이기 위해서.
Q. CUDA Stream은? → Inference와 memory copy를 병렬로 수행합니다.

H. Depth Anything

Q. Depth Anything은 왜 구현했습니까?

> Foundation Model 기반 Depth Estimation 구조를 이해하고 실제 프로젝트에 적용하기 위해 scratch부터 구현했습니다.

Q. Relative Depth란? → 절대 거리(mm)가 아니라 깊이의 상대적 순서(가까운지 먼지)만 아는 것.
Q. Metric Depth는? → 실제 거리(m).
Q. 왜 Metric으로 바꾸려 했나? → Robot Navigation에는 실제 거리가 필요하기 때문입니다.
Q. Monocular Depth의 한계는? → Scale Ambiguity가 존재합니다.

I. SLAM

Q. Voxelization은 왜 했습니까? → Point cloud 양을 줄여 계산량을 감소시키기 위해서.
Q. Octomap 장점? → Memory efficient · Occupancy 관리.

J. Sensor (센서 선정 경험 = CV의 강점)

Q. 센서 선정 시 가장 중요한 것은? → FOV · FPS · Latency · Interface · Power · Weight.
Q. Global Shutter가 필요한 이유? → 고속 이동 시 Motion Blur와 Rolling Shutter 문제를 줄일 수 있기 때문입니다.

K. 저해상도 / 작은 객체 Detection ★ (공고 핵심)

Q. 드론에서 작은 객체를 어떻게 Detection?

> Detector를 매 프레임 돌리지 않고, Tracker와 ROI 기반 Detection을 함께 쓰겠습니다. 저해상도 문제는 단일 프레임 확대로 풀리지 않으므로 foundation prior와 시간축 정보를 융합합니다.

Q. Motion Blur 해결? → Fast shutter · Global shutter · IMU fusion.
Q. Jetson이면? → TensorRT · FP16 · Async Pipeline · Zero Copy.

L. Sim-to-Real ★

Q. 시뮬레이션은 되는데 실제 드론에서 실패합니다. 무엇부터 확인? ★★ (최우선 암기)

> 순서대로 확인합니다.
> ① **Sensor Calibration** — IMU bias, camera intrinsic/extrinsic
> ② **Time Synchronization** — camera/IMU timestamp, delay
> ③ **Action Scaling** — 시뮬 모터 vs 실제 ESC 차이
> ④ **Dynamics** — mass, inertia, drag, propeller
> ⑤ **Observation Distribution** — 실제 데이터가 학습 분포와 다른지(OOD)

Q. Sim-to-Real Gap을 줄이는 방법? → Domain Randomization.
Q. 무엇을 Randomization? → Wind · Lighting · Mass · Friction · IMU noise · Camera noise · Motor delay · Battery voltage.

M. Safety Layer / Fallback ★ (공고에서 매우 중요)

Q. RL이 이상한 Action을 내면?

> RL 출력을 그대로 모터에 보내지 않고 Safety Filter를 거칩니다: `RL → Safety Filter → Controller → Motor`.
> Safety Filter에서 Velocity Limit · Geofence · Collision Check · Action Clipping을 수행합니다.

Q. Fallback은 언제? → Tracking 실패 · Sensor Error · Confidence 감소 · Latency 증가 시 PID 또는 MPC로 전환합니다.

N. HIL

Q. HIL을 왜 합니까? → 실제 hardware를 연결해 simulation과 실제의 차이를 줄이기 위해서.
Q. Edge Case는? → Camera Drop · GPS Loss · Wind · Low Battery.

O. CTO 경험

Q. 프로젝트를 어떻게 관리했습니까? → Jira · Sprint · Milestone · GitHub Integration.
Q. 기술 의사결정은? → POC를 먼저 만들고 성능과 유지보수성을 함께 고려했습니다.

P. 시스템 설계 (최종면접 수준) ★

Q. Jetson 하나로 Detection·Tracking·RL·PX4를 다 돌려야 합니다. 설계해보세요. ★★

```
Camera
  → Image Preprocessing
  → Detector (10Hz)          ← 비용 크므로 저주기
  → Tracker (30Hz)           ← 고주기
  → State Estimation
  → RL Policy (20Hz)         ← 상태 기반 상위 정책
  → Safety Filter
  → PX4 Position Controller
  → Attitude Controller
  → Motor                    ← 저수준 제어는 PX4 담당
```

함께 언급: **TensorRT FP16 · Detector/Tracker 분리 · Async Pipeline · Zero-copy Memory · CUDA Stream · ROI Detection · IMU-Camera Time Sync · Fallback Controller.**

Q. 입사하면 어떤 구조를 만들겠습니까?

> 먼저 시뮬레이션에서 재현 가능한 실험 환경을 구축하겠습니다. 모든 실험은 Config와 Seed를 저장해 재현성을 확보하고, Domain Randomization으로 Sim-to-Real Gap을 줄이겠습니다. 실제 드론에서는 RL이 직접 모터를 제어하지 않고 PX4 위에 Safety Layer를 두겠습니다. Vision은 Detector와 Tracker를 분리하고 Jetson에서는 TensorRT FP16 기반으로 최적화하겠습니다. 마지막으로 HIL 테스트로 실제 환경에서 정책을 단계적으로 검증하는 개발 프로세스를 구축하겠습니다.

Q. 최우선 암기 5 + 심층 토론 10

가장 먼저 암기할 답변 5개

1. **PPO vs SAC** — On-policy vs Off-policy, 안정성 vs Sample Efficiency.
2. **Sim은 되는데 실제는 안 되는 이유** — 센서 → 시간동기화 → Action scaling → 동역학 → 분포 차이 순서 점검.
3. **왜 RL이 모터를 직접 제어하면 안 되나** — 안전성·검증·설명 가능성. RL=상위 정책, PX4/PID/MPC=하위 제어.
4. **Jetson 최적화** — PyTorch → ONNX → TensorRT(FP16) → 프로파일링 → CPU/GPU 병목 제거 → Zero-copy·CUDA Stream.
5. **드론 전체 아키텍처** — Detection(저주기)·Tracking(고주기)·State Estimation·RL Policy·Safety Filter·PX4로 역할 분리.

10분 이상 토론 가능성 높은 주제 10개 (충분히 준비)

1. PPO/SAC/TD3 비교 및 드론 제어 적용
2. Sim-to-Real과 Domain Randomization
3. RT-DETR + TensorRT + Jetson 배포 경험
4. Tracking 구조 (Kalman, Hungarian, ID Switch)
5. Transformer 기반 Quadruped Learning
6. Video 기반 Imitation Learning과 Inverse Kinematics
7. Depth Anything와 Metric Depth 변환
8. Safety Layer, Fallback Controller, HIL 테스트 전략
9. Jetson 최적화 (ONNX, FP16, INT8, TensorRT, Zero-copy)
10. "시뮬은 되는데 실제 드론에서 실패하면 무엇부터 확인?" — 이 포지션에서 가장 가능성 높은 시스템 설계 질문.