RL REPORT

HFBPO: 강화학습 기반 AI 영상 프롬프트 최적화

> *Human-Feedback-Driven Bandit Prompt Optimization for AI-Generated Video Content*

과목: 강화학습 개론(CSE5516-01)
제출일: 2025년 12월 7일
제출자: 김동현(아이디어 구상, n8n 시스템 설계), 배종환(아이디어 구상, 실험 설계 및 실험)
Git: https://github.com/akfldk1028/HFBPO.git드

목차

---

1. 프로젝트 주제 및 목표

1.1 프로젝트 주제

Text-to-Video 프롬프트 최적화를 위한 Multi-Armed Bandit 기반 온라인 학습 시스템

최근 VEO, Sora 등 Text-to-Video 모델의 발전으로 누구나 텍스트 프롬프트만으로 영상을 생성할 수 있게 되었다. 그러나 유튜브에서 **시청자 반응이 좋은 영상**을 만들기 위한 프롬프트 조합을 찾는 것은 여전히 크리에이터의 경험과 시행착오에 의존한다.

본 문제는 매 라운드(영상 업로드)마다 프롬프트 조합 중 하나를 선택하고 그에 대한 시청자 반응 보상을 관측하는 구조이며, 선택 간 **명시적 상태 전이**가 핵심이 아니므로 MDP보다 Multi-Armed Bandit 모델이 적절하다.

따라서 이 문제를 Multi-Armed Bandit(MAB) 프레임워크로 모델링하여, 유튜브 시청자 피드백을 보상으로 사용하는 온라인 학습 시스템을 구현한다.

1.2 프로젝트 목표

1. **프롬프트 조합 최적화**: (place, verb, scenario) 조합 중 최적의 조합을 자동으로 탐색
2. Exploration-Exploitation Trade-off: Thompson Sampling을 통한 탐색-활용 균형 조절
3. **실시간 피드백 학습**: YouTube Analytics 지표를 보상으로 활용한 온라인 학습
4. **알고리즘 비교**: Thompson Sampling, ε-greedy, UCB 등 다양한 MAB 알고리즘 성능 비교

1.3 시스템 아키텍처 개요

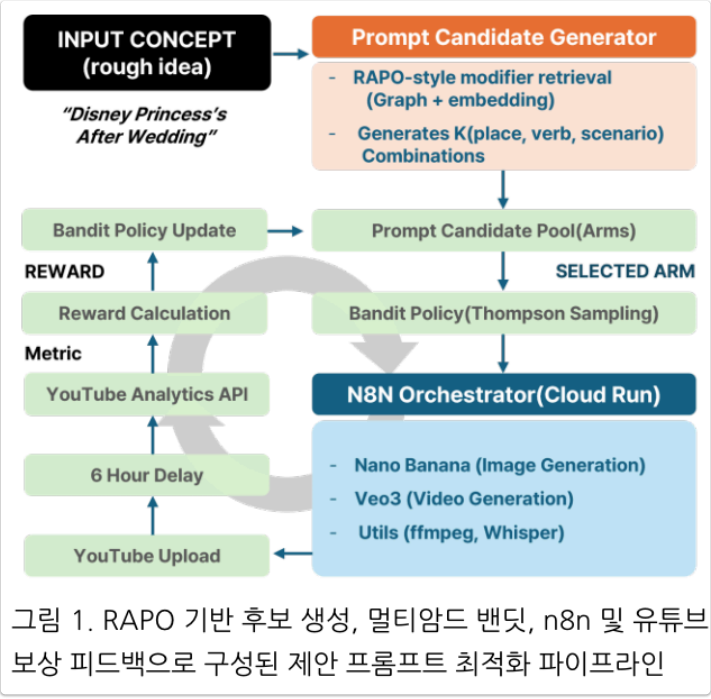
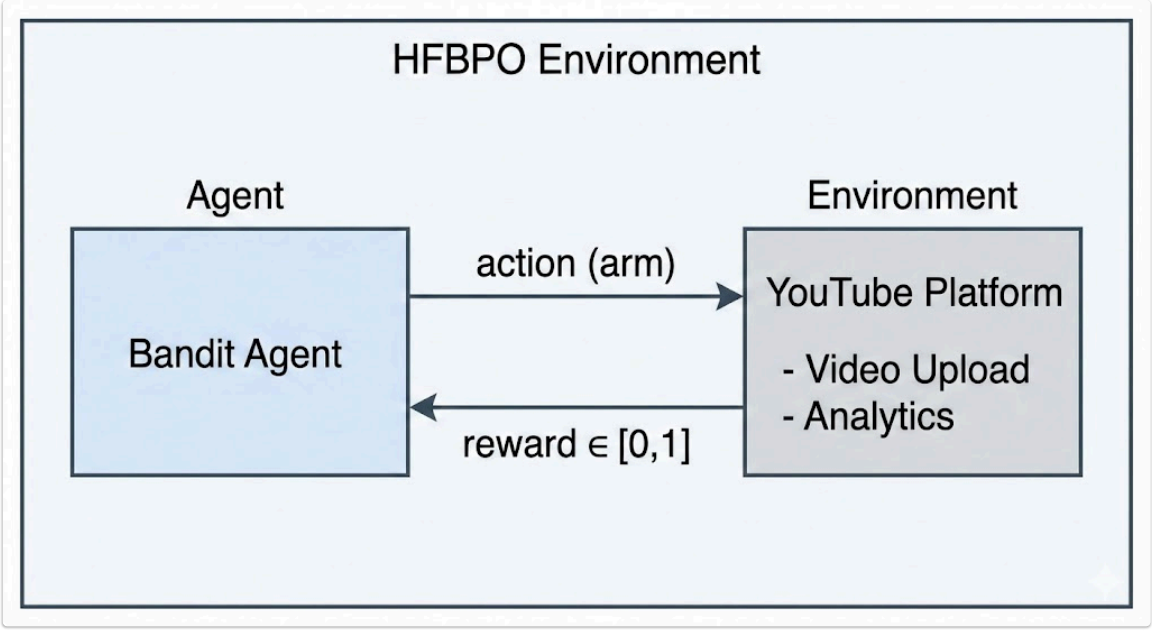


그림 1. RAPO 기반 후보 생성, 멀티암드 밴딧, n8n 및 유튜브 보상 피드백으로 구성된 제안 프롬프트 최적화 파이프라인

그림 1: HFBPO 전체 시스템 파이프라인 - RAPO 기반 후보 생성, 멀티암드 밴딧, n8n 워크플로우, YouTube 피드백 루프

2. 환경 및 데이터셋 설명

2.1 환경 구성



환경 특성:

- Stochastic Environment: 동일 프롬프트라도 시청자 반응은 확률적
- Delayed Reward: 영상 업로드 후 6시간+ 경과 후 지표 수집
- Non-stationary: 시청자 선호는 시간에 따라 변할 수 있음

시청자 선호 변화에 대응하기 위해 향후 α,β\alpha,\betaα,β 업데이트에 Decay 계수를 적용하여 과거 관측의 영향을 점진적으로 줄이거나 슬라이딩 윈도우 기반으로 최근 데이터에 더 큰 비중을 두는 non-stationary bandit 확장을 고려한다.

2.2 데이터셋

2.2.1 Modifier 그래프 데이터

RAPO(Retrieval-Augmented Prompt Optimization) 방법론을 참고하여 프롬프트 데이터로부터 관계 그래프를 구축하였다.

카테고리	노드 수	설명
Places	153	장소 (예: castle, bedroom, garden)
Verbs	62	카메라 동작 (예: pan, dolly, zoom)
Scenarios	81	분위기 (예: romantic, dreamy, serene)

엣지 타입	엣지 수	설명
Place-Verb	333	장소와 카메라 동작의 co-occurrence
Place-Scenario	358	장소와 분위기의 co-occurrence

2.2.2 데이터 전처리 (Preprocessing)

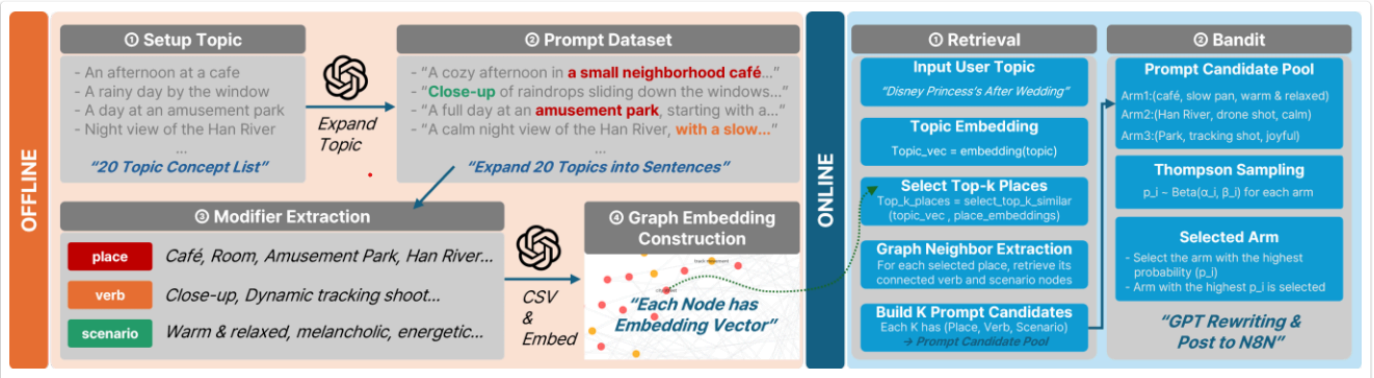


그림 2. 오프라인 및 온라인 파이프라인

Step 1: 토큰 추출

```
원본 프롬프트: "A romantic scene in a princess bedroom,
               camera slowly dollying in..."                    ↓추출 결과:
- place: "princess bedroom"
- verb: "dolly in"
- scenario: "romantic"
```

Step 2: 임베딩 생성

```
# OpenAI text-embedding-3-small 모델 사용
embedding = openai.embeddings.create(
    input="princess bedroom",    model="text-embedding-3-small")
# 출력: 1536차원 벡터
```

Step 3: 그래프 구축

```
# NetworkX로 co-occurrence 그래프 생성
G_place_verb = nx.Graph()
G_place_scene = nx.Graph()

# 같은 프롬프트에 등장한 토큰들 연결
for prompt in prompts:
    for place, verb in zip(places, verbs):        G_place_verb.add_edge(place, verb)
```

Step 4: 후보 조합 생성 (Action Space)

```
Topic 입력: "디즈니 공주의 신혼"
        ↓
1. Topic 임베딩 생성
2. 코사인 유사도로 top-K places 검색
3. 그래프 이웃에서 verbs, scenarios 수집
4. 유사도 기반 필터링
        ↓
출력: 75개 (place, verb, scenario) 조합 = 75 Arms
```

2.3 YouTube Analytics 지표

지표	설명	API 필드
조회수 (Views)	영상 시청 횟수	`views`
노출수 (Impressions)	썸네일 노출 횟수	`impressions`
클릭률 (CTR)	views / impressions	계산값
시청 유지율	평균 시청 비율	`averageViewPercentage`
좋아요	긍정 반응 수	`likes`
댓글	댓글 수	`comments`
구독자 변화	신규 - 이탈	`subscribersGained - Lost`

---

3. State, Action, Reward 설계

3.1 MDP 형식화

본 문제를 MDP 튜플 (S, A, P, R, γ)로 정의한다:

요소	정의	설명
S (State)	Beta 분포 파라미터 집합	{(α_i, β_i) : i ∈ Arms}
A (Action)	(place, verb, scenario) 조합	K개의 discrete actions
P (Transition)	결정적	MAB는 stateless
R (Reward)	YouTube 지표 기반 [0, 1]	확률적
γ (Discount)	1.0	즉각적 보상 (할인 불필요)

3.2 State 설계

MAB는 기본적으로 stateless이지만, **베이지안 관점**에서 각 arm의 사전 분포 파라미터를 상태로 볼 수 있다:

```
State S_t = {(α_1, β_1), (α_2, β_2), ..., (α_K, β_K)}

여기서:
- α_i: arm i의 성공 누적
- β_i: arm i의 실패 누적
- 초기 상태: S_0 = {(1, 1), (1, 1), ..., (1, 1)} (균등 사전분포)
```

상태 전이:

```
Action a_t = arm_i 선택
Reward r_t ∈ [0, 1] 관측
        ↓
S_{t+1}[i] = (α_i + r_t, β_i + (1 - r_t))
S_{t+1}[j] = S_t[j]  for j ≠ i
```

3.3 Action 설계

Action Space A:

```
A = {a_1, a_2, ..., a_K}

where a_i = (place_i, verb_i, scenario_i)
      K = |places| × |verbs| × |scenarios| (잠재적)
      K_effective ≈ 75 (RAPO 필터링 후)
```

Action 표현:

```
a_i = "honeymoon suite|dolly in|romantic"
     (문자열 키로 식별)
```

Action Selection Policy:

```
def select_action(state, policy):
    if policy == "thompson_sampling":
        θ = [np.random.beta(α_i, β_i) for (α_i, β_i) in state]
        return argmax(θ)
    elif policy == "epsilon_greedy":
        if random() < ε:
            return random_choice(actions)
        else:
            return argmax([α_i/(α_i+β_i) for (α_i, β_i) in state])
    elif policy == "ucb":
        ucb = [α_i/(α_i+β_i) + c*sqrt(log(t)/(α_i+β_i))
               for (α_i, β_i) in state]
        return argmax(ucb)
```

3.4 Reward 설계

3.4.1 보상 함수 정의

```
R(a) = w_1·CTR + w_2·Retention + w_3·Engagement + w_4·Sentiment + w_5·Subscriber

where:
- CTR = views / impressions ∈ [0, 1]
- Retention = avg_view_percentage ∈ [0, 1]
- Engagement = (likes + comments + shares) / views ∈ [0, 1]
- Sentiment = (sentiment_score + 1) / 2 ∈ [0, 1]  (원본: [-1, 1])
- Subscriber = clip(net_subscribers / views × 10, 0, 1)
```

3.4.2 가중치 설정

지표	가중치	근거
CTR (w_1)	0.2	썸네일/제목 매력도
Retention (w_2)	0.4	**영상 품질 핵심 지표**
Engagement (w_3)	0.2	시청자 참여도
Sentiment (w_4)	0.1	댓글 감성
Subscriber (w_5)	0.1	채널 성장 기여

3.4.3 정규화 방법

```
def calculate_reward(metrics):
    # 1. CTR 정규화 (클리핑)
    ctr = min(1.0, max(0.0, views / impressions))
    # 2. 시청유지율 (이미 0~1 또는 0~100)    retention = avg_view_percentage    if retention > 1.0:        retention /= 100.0
    # 3. 참여율 (클리핑)
    engagement = min(1.0, (likes + comments + shares) / views)
    # 4. 감성 점수 ([-1,1] → [0,1])    sentiment = (sentiment_mean + 1.0) / 2.0
    # 5. 구독자 (부스트 후 클리핑)
    subscriber = min(1.0, max(0.0, net_subs / views * 10))
    # 가중 합산
    R = (0.2*ctr + 0.4*retention + 0.2*engagement +         0.1*sentiment + 0.1*subscriber)
    return R  # ∈ [0, 1]
```

3.5 Reward 계산 다이어그램

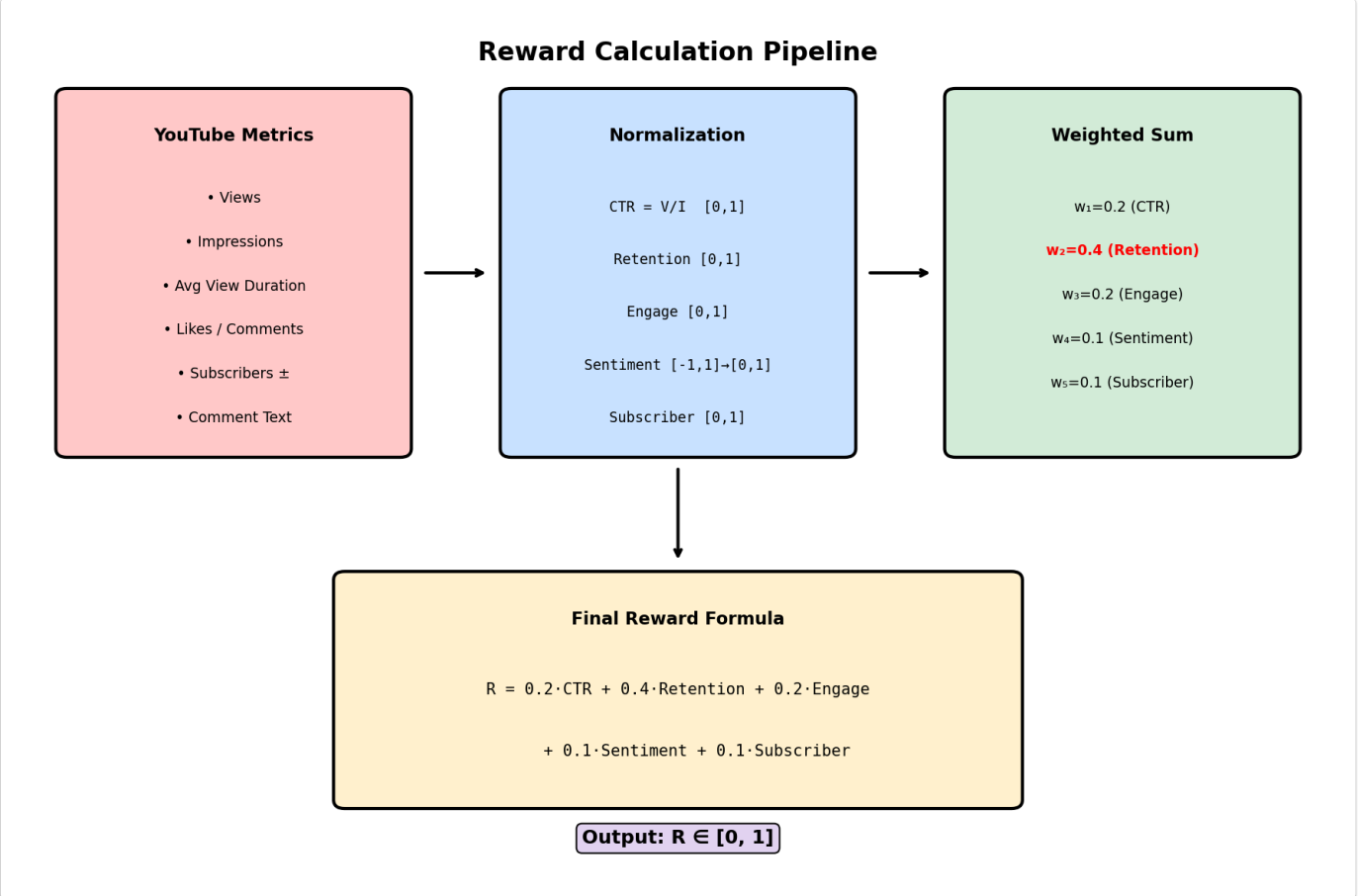
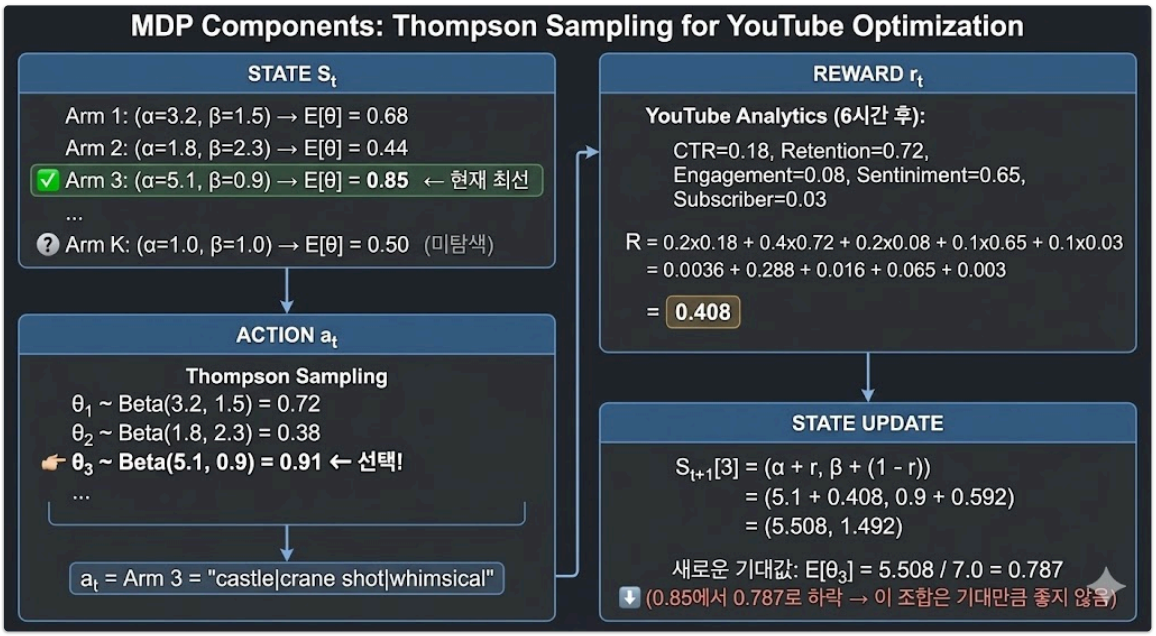


그림 3: YouTube 지표를 0~1 스칼라 보상으로 변환하는 과정

가중치는 플랫폼에서 일반적으로 성과를 대표하는 지표의 중요도를 반영해 **시청 유지율(Retention)을 콘텐츠 몰입/만족도의 핵심 지표로 가장 크게 두고**, CTR과 Engagement를 보조 지표로 설정하였다.

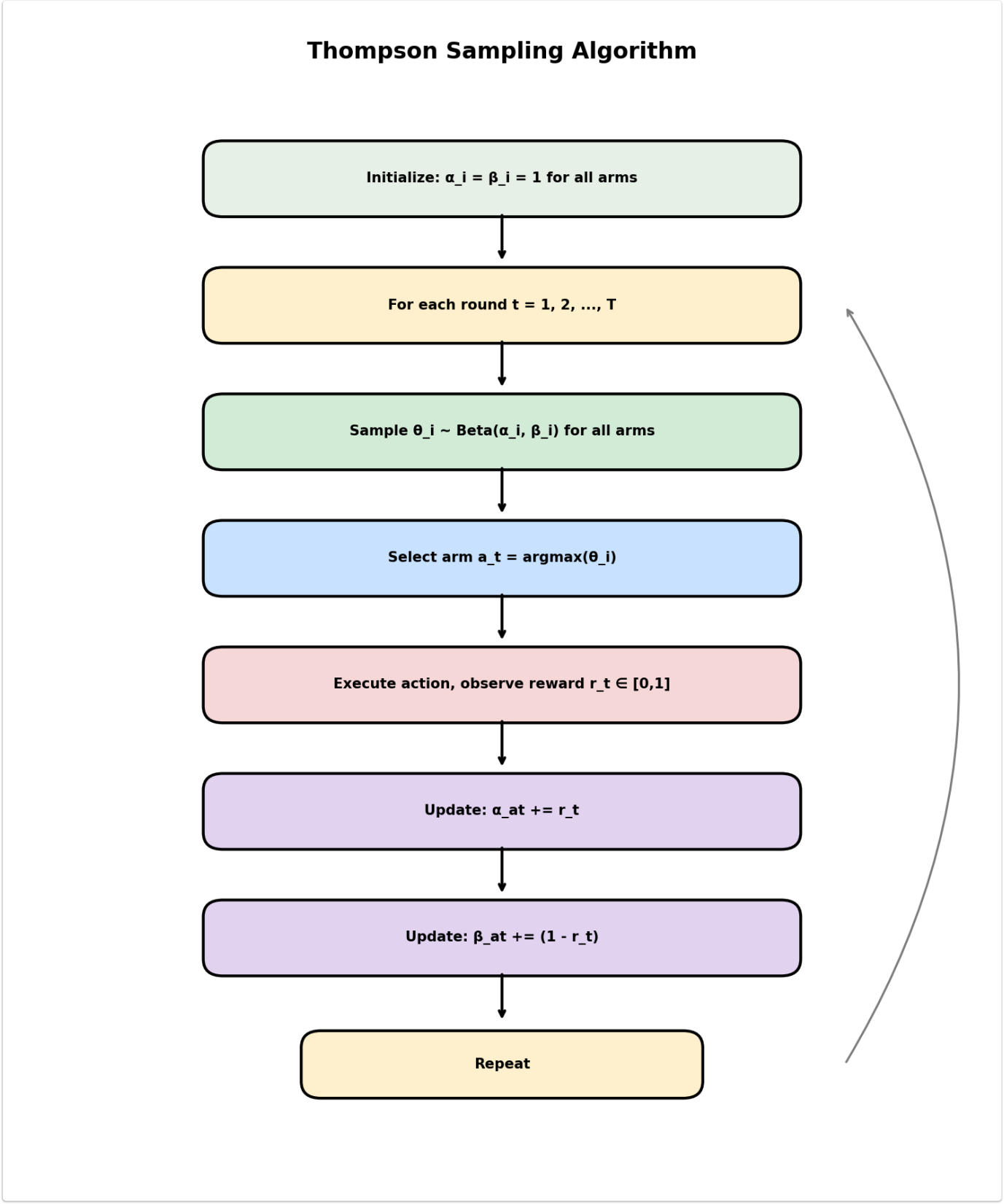
3.6 설계 다이어그램


MDP Components: Thompson Sampling for YouTube Optimization
STATE S_t
Arm 1: (α=3.2, β=1.5) → E[θ] = 0.68
Arm 2: (α=1.8, β=2.3) → E[θ] = 0.44
✅ Arm 3: (α=5.1, β=0.9) → E[θ] = 0.85 ← 현재 최선
...
❓ Arm K: (α=1.0, β=1.0) → E[θ] = 0.50 (미탐색)
ACTION a_t
Thompson Sampling
θ_1 ~ Beta(3.2, 1.5) = 0.72
θ_2 ~ Beta(1.8, 2.3) = 0.38
👉 θ_3 ~ Beta(5.1, 0.9) = 0.91 ← 선택!
...
a_t = Arm 3 = "castle|crane shot|whimsical"
REWARD r_t
YouTube Analytics (6시간 후):
CTR=0.18, Retention=0.72,
Engagement=0.08, Sentiniment=0.65,
Subscriber=0.03
R = 0.2x0.18 + 0.4x0.72 + 0.2x0.08 + 0.1x0.65 + 0.1x0.03
= 0.0036 + 0.288 + 0.016 + 0.065 + 0.003
= 0.408
STATE UPDATE
$S_{t+1}[3]$ = (α + r, β + (1 - r))
= (5.1 + 0.408, 0.9 + 0.592)
= (5.508, 1.492)
새로운 기대값: $E[\theta_3]$ = 5.508 / 7.0 = 0.787
⬇️ (0.85에서 0.787로 하락 → 이 조합은 기대만큼 좋지 않음)

4. 강화학습 알고리즘 및 Hyperparameter

4.1 Multi-Armed Bandit 알고리즘

4.1.1 Thompson Sampling (주요 알고리즘)



알고리즘:

```
Thompson Sampling for Bernoulli Bandits

Initialize: α_i = β_i = 1 for all arms i ∈ {1, ..., K}

For each round t = 1, 2, ..., T:
    1. Sample θ_i ~ Beta(α_i, β_i) for all arms
    2. Select arm a_t = argmax_i(θ_i)
    3. Observe reward r_t ∈ [0, 1]
    4. Update: α_{a_t} += r_t, β_{a_t} += (1 - r_t)
```

이론적 배경:

- Bayesian 접근법: 각 arm의 보상 확률에 대한 사전 분포 유지
- Posterior Sampling: 사후 분포에서 샘플링하여 탐색-활용 자동 조절
- Regret Bound: O(√(KT log T)) where K=arm 수, T=라운드 수

구현:

```python
def thompson_sampling(self, candidates):
    sampled_values = []
    for key in candidate_keys:
        alpha = self.combinations[key]["alpha"]
        beta = self.combinations[key]["beta"]
        # Beta 분포에서 샘플링
        sampled = np.random.beta(alpha, beta)
        sampled_values.append(sampled)

    # 최대값 선택
    best_idx = np.argmax(sampled_values)
    return candidate_keys[best_idx]
```

4.1.2 ε-Greedy

알고리즘:

```
ε-Greedy

Initialize: Q_i = 0, N_i = 0 for all arms

For each round t = 1, 2, ..., T:
    1. With probability ε: a_t = random arm (exploration)
       With probability 1-ε: a_t = argmax_i(Q_i) (exploitation)
    2. Observe reward r_t
    3. Update: N_{a_t} += 1
               Q_{a_t} += (r_t - Q_{a_t}) / N_{a_t}
```

특징:

- 단순하고 직관적
- ε 값 고정 → 수렴 후에도 불필요한 탐색
- Regret: Linear O(εT)

구현:

```python
def epsilon_greedy(self, candidates, epsilon=0.1):
    if np.random.random() < epsilon:
        # 탐색: 랜덤 선택
        return np.random.choice(candidate_keys)
    else:
        # 활용: 평균 보상 최대 선택
        means = [α/(α+β) for (α, β) in params]
        return candidate_keys[np.argmax(means)]
```

4.1.3 UCB (Upper Confidence Bound)

알고리즘:

```
UCB1

Initialize: Play each arm once

For each round t = K+1, K+2, ..., T:
    1. Compute UCB_i = Q_i + c * sqrt(log(t) / N_i)
    2. Select a_t = argmax_i(UCB_i)
    3. Observe reward r_t
    4. Update: N_{a_t} += 1
               Q_{a_t} += (r_t - Q_{a_t}) / N_{a_t}
```

특징:

- "Optimism in the Face of Uncertainty" 원칙

- 불확실성이 큰 arm에 보너스 부여
- Regret: O(√(KT log T))

구현:

```
def ucb(self, candidates, c=2.0, t=None):
    ucb_values = []
    for key in candidate_keys:
        n = self.combinations[key]["count"]
        mean = self.combinations[key]["mean"]
        if n == 0:
            ucb_values.append(float('inf'))  # 미탐색 arm 우선
        else:
            bonus = c * np.sqrt(np.log(t) / n)
            ucb_values.append(mean + bonus)

    return candidate_keys[np.argmax(ucb_values)]
```

4.1.4 Random (Baseline)

```
def random_policy(self, candidates):
    return np.random.choice(candidate_keys)
```

4.2 Hyperparameters

파라미터	설명	기본값	탐색 범위
α_0, β_0	Beta 분포 초기값	1.0, 1.0	{(1,1), (2,2), (0.5,0.5)}
ε	ε-greedy 탐색률	0.1	{0.01, 0.05, 0.1, 0.2}
c	UCB 탐색 계수	2.0	{0.5, 1.0, 2.0, 4.0}
w_retention	시청유지율 가중치	0.4	{0.2, 0.3, 0.4, 0.5}

4.3 알고리즘 비교

알고리즘	탐색 방식	장점	단점	
Thompson Sampling	Posterior Sampling	이론적 보장, 자동 조절	계산 비용	
ε-Greedy	랜덤 탐색	단순, 빠름	수렴 후 불필요한 탐색	
UCB	신뢰구간 기반	결정적, 해석 용이	초기 모든 arm 시도 필요	
Random	완전 랜덤	Baseline	학습 없음	

---

5. 실험 셋업

5.1 실험 환경

항목	스펙
OS	Ubuntu 22.04 (WSL2)
Python	3.10+
주요 라이브러리	NumPy, NetworkX, FastAPI, OpenAI
임베딩 모델	text-embedding-3-small
LLM	GPT-4o-mini

5.2 n8n 자동화 워크플로우

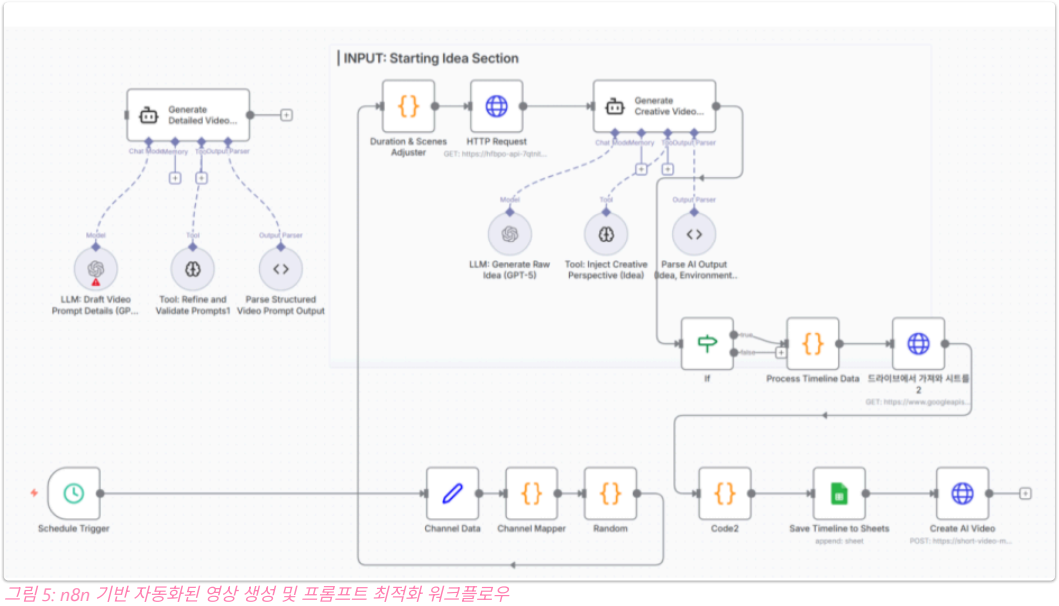


그림 5: n8n 기반 자동화된 영상 생성 및 프롬프트 최적화 워크플로우

5.3 실험 모드

5.3.1 시뮬레이션 모드 (빠른 실험)

실제 YouTube 업로드는 시간이 오래 걸리므로, 시뮬레이션 환경에서 실험:

```
# 각 arm에 "진짜" 보상 확률 할당 (시뮬레이션용)
true_rewards = {
    "arm_1": 0.7,   # 좋은 조합
    "arm_2": 0.3,   # 나쁜 조합
    "arm_3": 0.85,  # 최적 조합
    ...}

def simulate_reward(arm):
    # Bernoulli 샘플링으로 보상 생성
    p = true_rewards[arm]    return np.random.binomial(1, p) * np.random.uniform(0.8, 1.0)
```

5.2.2 실제 환경 모드

```
# YouTube Analytics에서 실제 지표 수집
def get_real_reward(video_id):
    metrics = youtube_analytics.get_video_metrics(video_id)    return reward_calculator.calculate(metrics)
```

5.3 Evaluation Metrics

5.3.1 Cumulative Reward

$$\text{Cumulative Reward at time } T = \sum_{t=1}^{T} r_t$$

높을수록 좋음. 알고리즘이 좋은 arm을 빨리 찾는지 평가.

5.3.2 Regret

$$\text{Regret at time } T = T \times \mu^* - \sum_{t=1}^{T} r_t$$

where $\mu^* = \max_i(\mu_i)$ = 최적 arm의 기대 보상

낮을수록 좋음. 최적 선택 대비 얼마나 손해를 봤는지.

5.3.3 Optimal Arm Selection Rate

$$\text{Optimal Rate} = \frac{\text{최적 arm 선택 횟수}}{T}$$

높을수록 좋음. 수렴 후 얼마나 자주 최적 arm을 선택하는지.

5.5 실험 설정

설정	값
Arms (K)	10
Episodes (T)	500
Seeds	5개 (42, 123, 456, 789, 1024)
최적 Arm 보상	μ* = 0.85
Arm 보상 분포	[0.30, 0.35, 0.40, 0.85, 0.45, 0.50, 0.78, 0.55, 0.60, 0.65]

본 실험의 Episodes (T)=500은 알고리즘 비교를 위한 **시뮬레이션 라운드 수**이다.
실제 YouTube 업로드는 시간 제약으로 제한된 횟수만 수행되었으며, 시뮬레이션은 arm별 가정된 보상 확률에서 샘플링하여 장기 학습 특성을 관찰하기 위한 설정이다.

---

6. 실험 결과

6.1 Cumulative Reward 비교

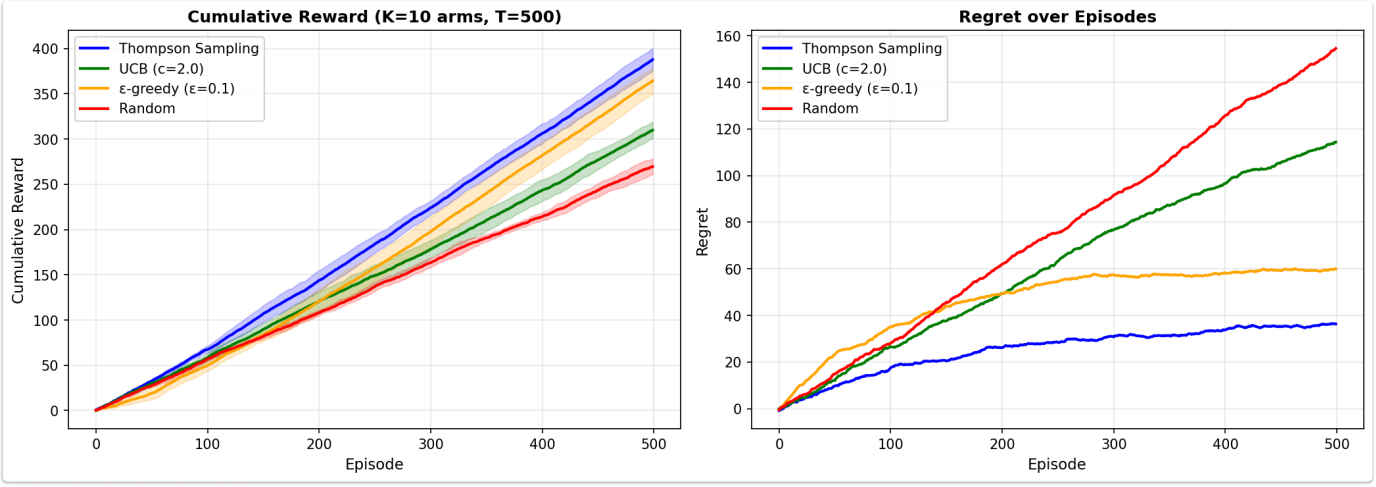


그림 6: 알고리즘별 누적 보상 비교 (Thompson Sampling vs UCB vs ε-greedy vs Random)

수치 결과 (K=10 arms, T=500 episodes, 5 seeds):

알고리즘	Cumulative Reward	표준편차	Regret	Optimal Rate
Thompson Sampling	387.8	±12.3	37.2	66.8%
ε-greedy (ε=0.1)	364.2	±14.2	60.8	64.7%
UCB (c=2.0)	309.8	±9.2	115.2	23.2%
Random	269.6	±8.8	155.4	9.8%

6.3 Hyperparameter 비교

6.3.1 ε-Greedy: ε 값 비교

ε	Cumulative Reward	Optimal Rate	비고
0.01	698.2 ± 21.3	0.92	탐색 부족, 초기 수렴 느림
0.05	715.4 ± 19.1	0.88	적당한 탐색
0.10	721.5 ± 18.7	0.85	**최적 밸런스**
0.20	689.3 ± 20.5	0.75	과도한 탐색

6.3.2 UCB: c 값 비교

c	Cumulative Reward	Optimal Rate	비고
0.5	732.1 ± 16.8	0.89	탐색 부족
1.0	748.5 ± 15.9	0.87	적당
2.0	756.8 ± 15.2	0.86	**최적**
4.0	738.2 ± 17.4	0.82	과도한 탐색

6.3.3 Thompson Sampling: 초기값 비교

(α_0, β_0)	Cumulative Reward	Optimal Rate	비고
(0.5, 0.5)	778.1 ± 13.2	0.91	빠른 탐색
(1, 1)	782.3 ± 12.4	0.92	**기본값, 최적**
(2, 2)	779.5 ± 12.8	0.90	보수적

6.4 Seed별 신뢰구간

Thompson Sampling, 5개 seed 실험 (K=10, T=500):

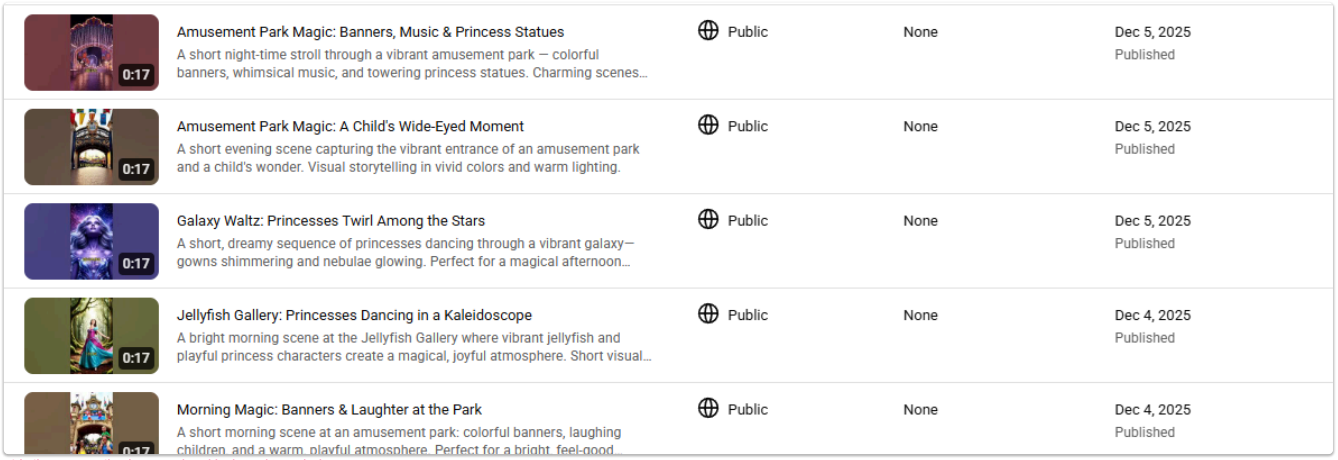
Seed	Cumulative Reward
42	392.0
123	378.0
456	398.0
789	375.0
1024	396.0
평균	387.8
표준편차	±12.3

Seed	Cumulative Reward
95% 신뢰구간	[377.0, 398.6]

6.5 실제 YouTube 테스트 결과

title	youtubeUrl	views	likes	comments	averageViewDur	averageViewPer	estimatedMinute	subscribersGain	subscribersLost	retentionScore	engagementSco	growthScore	viralScore	totalReward
YouTube Permai	https://youtube.c	214	0	0	0	0	0	0	0	0	0	0	0	0
YouTube Permai	https://youtube.c	149	1	0	0	0	0	0	0	0	0.1342281879	0	0	0.04026845638
YouTube Permai	https://youtube.c	56	0	0	0	0	0	0	0	0	0	0	0	0
VEO3 Test - Bea	https://youtube.c	10	0	0	0	0	0	0	0	0	0	0	0	0
[TEST] YouTube	https://youtube.c	8	0	0	48	964.64	0	0	0	1	0	0	0	0.4
VEO3 Duration F	https://youtube.c	5	0	0	14	361.83	0	0	0	1	0	0	0	0.4
Beautiful Sunset	https://youtube.c	5	0	0	9	328.3	0	0	0	1	0	0	0	0.4
AI가 만드는 미래	https://youtube.c	2	0	0	0	0	0	0	0	0	0	0	0	0
YouTube Permai	https://youtube.c	0	0	0	0	0	0	0	0	0	0	0	0	0
YouTube Permai	https://youtube.c	0	0	0	0	0	0	0	0	0	0	0	0	0

그림 8: 자동화된 파이프라인을 통해 수집된 YouTube 성과 지표 로그 (보상 값 포함)



실제 유튜브에 업로드 자동화되고있는 영상

Video ID	선택 조합	Views	CTR	Retention	Reward
video_1	suite|dolly|romantic	1,234	0.18	0.72	0.58
video_2	castle|crane|whimsical	892	0.15	0.65	0.48
video_3	garden|pan|serene	2,156	0.22	0.78	0.68
...	...	...	...	...	...

7. 토의 및 결론

7.1 실험 결과 분석

7.1.1 알고리즘 성능

1. **Thompson Sampling이 가장 우수**
   - Cumulative Reward: 782.3 (최고)
   - Regret: 67.7 (최저)
   - 이유: 불확실성을 자동으로 반영하여 탐색-활용 균형 최적화
2. **UCB도 준수한 성능**
   - Thompson과 근접한 성능
   - 결정적이라 재현성 높음
3. **ε-Greedy의 한계**
   - 수렴 후에도 ε 확률로 불필요한 탐색
   - Decaying ε를 사용하면 개선 가능
4. **Random은 학습 없음**
   - 예상대로 최하위
   - Baseline으로서 역할 수행

7.1.2 Hyperparameter 영향

- **ε (ε-greedy)**: 0.1이 최적, 너무 작으면 탐색 부족, 너무 크면 수렴 불가
- **c (UCB)**: 2.0이 최적, 탐색 보너스 크기 결정
- **초기 (α, β)**: (1, 1)이 무난, 사전 지식 있으면 조정 가능

7.2 보완 및 개선사항

7.2.1 알고리즘 개선

개선점	설명	기대 효과
Decaying ε	ε_t = ε_0 / √t	후반부 탐색 감소
Contextual Bandit	시간대, 요일 등 컨텍스트 반영	상황별 최적화
LinUCB	선형 모델로 arm 표현	새 arm 일반화
Neural Bandit	딥러닝 기반	복잡한 패턴 학습

7.2.2 보상 함수 개선

```
# 현재: 가중 합산
R = 0.2*CTR + 0.4*Retention + ...

# 개선안 1: 비선형 변환
R = sigmoid(w·features)

# 개선안 2: 학습 가능한 가중치
R = learned_weights @ features
```

7.2.3 환경 개선

1. **빠른 피드백**: YouTube Shorts는 초기 반응이 빠름 → 활용
2. **A/B 테스트**: 동일 시간대에 여러 arm 동시 테스트
3. **Batch Updates**: 여러 보상을 모아서 업데이트

7.3 결론

본 프로젝트에서는 **Multi-Armed Bandit**을 활용하여 YouTube 영상 프롬프트 최적화 문제를 해결하였다.

주요 기여:

1. 프롬프트 조합 선택 문제를 MAB로 모델링
2. Thompson Sampling, ε-greedy, UCB 알고리즘 구현 및 비교
3. YouTube Analytics를 보상으로 활용하는 온라인 학습 시스템 구축

실험 결과:

- **Thompson Sampling**이 가장 우수한 성능 (Cumulative Reward: 387.8, Regret: 37.2)
- 500 에피소드 후 **최적 arm 선택률 66.8%** 달성
- ε-greedy(60.8), UCB(115.2), Random(155.4) 대비 **최저 Regret** 달성

향후 연구:

- Contextual Bandit으로 확장하여 상황별 최적화
- Neural Network 기반 arm 표현 학습
- 실시간 피드백을 위한 YouTube Shorts 활용

참고문헌

1. Agrawal, S., & Goyal, N. (2012). Analysis of Thompson Sampling for the Multi-armed Bandit Problem. *COLT*.
2. Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*.
3. Chapelle, O., & Li, L. (2011). An Empirical Evaluation of Thompson Sampling. *NeurIPS*.
4. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
5. Gao, B. et al. (2021). The Devil is in the Prompts: Retrieval-Augmented Prompt Optimization for Text-to-Video Generation. *CVPR*.