협업 및 커뮤니케이션 컨벤션

회의 및 진행 방식

- **세션 기록**: 모든 스터디 세션을 녹화하거나 상세 노트 작성, 주요 결정사항과 액션 아이템 명시[1] [2]
- **타임박스 설정**: 각 발표는 15-20분 이내, Q&A는 10분 이내로 제한하여 효율성 확보
- **사전 자료 공유**: 발표 전날까지 슬라이드/코드를 공유하여 참석자가 미리 검토 가능하도록 설정
- **비동기 피드백**: GitHub Discussions나 Slack 채널에서 스터디 외 시간에도 질문/의견 교환
- **회고 세션**: 매월 말 프로세스 개선을 위한 회고 진행 (무엇이 잘 됐고, 개선할 점은?)

역할 및 책임

- **로테이션 시스템**: 매주 코드 리뷰어, 문서화 담당, 발표 진행자를 순환 배정[1]
- **명확한 오너십**: 각 RAG 컴포넌트마다 primary owner 지정 (예: 인덱싱-표제님)
- **백업 담당자**: 주 담당자가 불가시 대체 가능한 부담당자 설정

실험 및 평가 컨벤션

실험 추적 (Experiment Tracking)

- **실험 메타데이터 기록**: 모든 실험은 날짜, 담당자, 목적, 하이퍼파라미터, 결과를 표준화된 양식으로 기록[3] [4]
- **재현 가능성**: 실험 코드는 반드시 requirements.txt (또는 uv.lock) + seed 값 고정으로 재현 가능하도록 작성
- **실험 브랜치 네이밍**: `exp/{담당자명}/{실험내용}` 형식 (예: `exp/kim/bge-m3-finetuning`)
- **실험 로그 도구**: Weights & Biases, MLflow, 또는 간단한 CSV 로그 중 하나 선택하여 일관되게 사용[5]

평가 기준

- **정량 지표**: Retrieval Precision@K, Recall@K, MRR, NDCG 등 객관적 지표 우선[6] [7]
- **정성 평가**: 매주 5-10개 샘플 쿼리에 대해 수동 평가 (관련성, 정확성, 완전성)[7] [8]
- **골드 데이터셋**: 랩큐 도메인 특화 평가셋 구축 (초기 20-30개 질문-답변 쌍, 점진적 확장)
- **개선 임계값**: 최소 5% 이상 성능 향상시에만 merge 고려 (통계적 유의성 확인)

문서화 컨벤션

코드 문서화

- **Docstring 필수**: 모든 public 함수/클래스에 Google/NumPy 스타일 docstring 작성[9]
- **타입 힌팅**: Python 3.12 타입 힌트 적극 활용 (mypy 검사 통과)
- **인라인 주석**: 복잡한 로직에만 "왜(why)"를 설명하는 주석 추가 ("무엇(what)"은 코드로 설명)

프로젝트 문서화

- **README 구조화**: 프로젝트 개요, 설치 방법, 빠른 시작, 아키텍처 다이어그램, 기여 가이드 포함
- CHANGELOG.md: 매 스프린트/주차별 주요 변경사항 기록[10]
- **ADR (Architecture Decision Records)**: 중요한 기술적 결정은 문서화 (왜 Qdrant를 선택했는지, 왜 BGE-M3인지 등)
- **실험 결과 문서**: `/docs/experiments/` 폴더에 주차별 실험 결과 마크다운 정리

지식 공유

- **학습 노트**: 각자 공부한 RAG 요소는 `/docs/learning/` 에 간략한 요약 + 참고 링크 정리
- **용어집**: 팀 내 공통 용어 정의 (chunking, embedding, retrieval, reranking 등)
- **템플릿 제공**: PR 템플릿, 이슈 템플릿, 실험 보고서 템플릿 사전 정의

코드 품질 컨벤션

코드 스타일

- **Formatter**: Black (line-length=100) + isort 적용, pre-commit hook 설정[5]
- **Linter**: Ruff 또는 Pylint 사용, 최소 8.0/10 점수 유지
- **네이밍**: 변수/함수는 snake_case, 클래스는 PascalCase, 상수는 UPPER_CASE
- **파일 구조**: `/src/{모듈명}/`, `/tests/`, `/scripts/`, `/notebooks/` 명확히 분리

코드 리뷰

- **리뷰 타임라인**: PR 생성 후 24시간 이내 1차 리뷰, 48시간 이내 approve/request changes[1]
- **셀프 리뷰**: PR 생성 전 자신의 코드를 먼저 검토하고 체크리스트 확인
- **건설적 피드백**: "이건 왜 이렇게 했나요?" 대신 "이 부분을 X 방식으로 하면 Y 장점이 있을 것 같습니다" 형식
- **필수 리뷰어**: 최소 1명 approve 필수, 핵심 변경은 2명 approve

테스팅

- **단위 테스트**: 핵심 로직 (parser, splitter, retriever)에 pytest 단위 테스트 작성
- **통합 테스트**: End-to-end 파이프라인 테스트 (문서 입력 → 답변 생성)
- **테스트 커버리지**: 최소 60% 목표, 점진적으로 80% 달성
- **CI/CD**: GitHub Actions로 PR시 자동 테스트 실행

데이터 관리 컨벤션

데이터 버전 관리

- **DVC 사용 고려**: 문서 데이터셋이 크면 DVC로 버전 관리[5]
- **데이터 스냅샷**: 주요 실험 전 데이터 상태 스냅샷 저장 (날짜 태그)
- **데이터 출처 기록**: 노션/구글드라이브 문서의 수집 일자, 전처리 이력 메타데이터 관리

데이터 품질

- **정기 재인덱싱**: 원본 문서 업데이트시 즉시 재인덱싱 트리거[11]
- **데이터 검증**: 수집된 문서의 포맷, 중복, 누락 체크 스크립트
- **개인정보 필터링**: 랩큐 내부 민감 정보 자동 마스킹/제거 프로세스

인프라 및 배포 컨벤션

환경 관리

- **환경 분리**: `dev`, `staging`, `prod` 환경 명확히 구분 (초기엔 dev만)
- **환경 변수**: `.env.example` 제공, 실제 `.env`는 gitignore, 민감 정보는 절대 커밋 금지
- **Docker 컴포즈**: Qdrant + API 서버 + 프론트엔드를 docker-compose로 일관된 환경 구성
- **의존성 고정**: uv로 lock 파일 관리, 라이브러리 업데이트는 별도 PR로 분리

모니터링 및 로깅

- **구조화된 로깅**: JSON 형식 로그, 타임스탬프 + 로그 레벨 + 메시지 + 컨텍스트 포함
- **성능 모니터링**: 쿼리 응답 시간, 임베딩 시간, 검색 시간 등 주요 메트릭 추적[8] [12]
- **에러 트래킹**: Sentry나 간단한 에러 로그 수집으로 프로덕션 이슈 파악
- **대시보드**: Grafana나 Streamlit 대시보드로 주요 지표 시각화

학습 및 성장 컨벤션

지식 확산

- **페어 프로그래밍**: 복잡한 기능은 2인 페어로 개발 고려
- **코드 워크스루**: 매주 한 명이 자신의 구현을 라이브 코딩으로 설명
- **논문/블로그 공유**: RAG 관련 최신 논문/기술 블로그는 Slack에 공유 + 간단 요약
- **외부 발표 권장**: 학회/밋업에서 프로젝트 발표로 피드백 수집

문제 해결

- **이슈 트래커 활용**: 막힌 부분은 즉시 GitHub Issue로 등록, 라벨링 (bug/question/enhancement)
- **타임박스 디버깅**: 30분 이상 막히면 팀원에게 도움 요청 (비효율 방지)
- **포스트모템**: 큰 버그/장애 발생시 원인 분석 + 재발 방지책 문서화

윤리 및 보안 컨벤션

데이터 프라이버시

- **접근 권한**: 랩큐 내부 데이터 접근 권한 명확히 관리, 외부 공유 금지[8]
- **익명화**: 평가/공유용 데이터는 개인정보 익명화 처리
- **규정 준수**: 회사 보안 정책 준수, 민감 정보 처리 가이드라인 숙지

AI 윤리

- **편향 모니터링**: 생성 결과의 편향성 주기적 점검 (성별, 직급 등)[8]
- **투명성**: 답변 출처(retrieved documents) 사용자에게 제공
- **사용자 피드백**: 부정확한 답변 신고 기능, 피드백 루프 구축[8]

시간 및 일정 컨벤션

스프린트 관리

- **주간 목표**: 매주 목요일 스터디에서 다음 주 목표 설정 (SMART 원칙)
- **진행률 체크**: 주중 수요일 비동기로 진행률 공유 (Slack 스탠드업)
- **버퍼 타임**: 예상 작업 시간의 1.5배로 계획 (불확실성 대비)

의사결정

- **합의 원칙**: 기술적 결정은 데이터 기반 + 팀 합의, 의견 불일치시 실험으로 검증
- **빠른 결정**: 사소한 결정(변수명 등)은 즉시, 중요한 결정(아키텍처)은 24시간 숙고
- **문서화된 결정**: 모든 중요 결정은 ADR이나 회의록에 기록

이 컨벤션들은 RAG 시스템 개발의 전 과정—협업, 실험, 코드 품질, 문서화, 배포, 학습—을 체계화하여 3인 스터디의 효율성과 학습 효과를 극대화합니다. 초기에는 핵심적인 몇 가지만 선택하고, 점진적으로 확장해나가는 것을 권장합니다.[11] [10] [1]

⁂

1. https://neptune.ai/blog/ml-collaboration-best-practices-from-ml-teams
2. https://dev.to/mohammad-reza-mahdiani/ai-powered-collaboration-how-machine-learning-can-transform-team-dynamics-fha
3. https://neptune.ai/blog/ml-experiment-tracking
4. https://viso.ai/deep-learning/experiment-tracking/
5. https://ml-ops.org/content/mlops-principles
6. https://pub.towardsai.net/i-spent-3-months-building-ra-systems-before-learning-these-11-strategies-1a8f6b4278aa
7. https://www.kapa.ai/blog/how-to-build-a-rag-pipeline-from-scratch-in-2026
8. https://marutitech.com/best-practices-for-rag-systems/
9. https://docs.aws.amazon.com/prescriptive-guidance/latest/writing-best-practices-rag/best-practices.html
10. https://www.aggil.fr/blog/rag-2025-best-practices
11. https://dev.to/pavanbelagatti/learn-how-to-build-reliable-rag-applications-in-2026-1b7p
12. https://redis.io/blog/rag-at-scale/
13. https://www.reddit.com/r/mlops/comments/1q87ytk/a_practical_2026_roadmap_for_modern_ai_search_rag/
14. https://go.comet.ml/rs/912-JJP-445/images/eBook-Comet-Building_effective_ML_teams.pdf
15. https://newxel.com/blog/building-a-machine-learning-team/