МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
МЭДЭЭЛЛИЙН ТЕХНОЛОГИ, ЭЛЕКТРОНИКИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРЫН УХААНЫ ТЭНХИМ

Булганы Раднаабазар

Микросервис архитектурт суурилсан хиймэл оюун агентууд

(AI agents for microservices)

Мэдээллийн технологи (D061304)
Дипломын ажлын тайлан

Улаанбаатар

2025 оны 12 сар

МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
МЭДЭЭЛЛИЙН ТЕХНОЛОГИ, ЭЛЕКТРОНИКИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРЫН УХААНЫ ТЭНХИМ

Микросервис архитектурт суурилсан хиймэл оюун агентууд

(AI agents for microservices)

Мэдээллийн технологи (D061304)
Дипломын ажлын тайлан

Удирдагч: _____ Дэд профессор Б.Сувдаа

Гүйцэтгэсэн: _____ Б.Раднаабазар (22B1NUM0286)

Улаанбаатар

2025 оны 12 сар

Зохиогчийн баталгаа

Миний бие Булганы Раднаабазар ”Микросервис архитектурт суурилсан хиймэл оюун агентууд” сэдэвтэй судалгааны ажлыг гүйцэтгэсэн болохыг зарлаж дараах зүйлсийг баталж байна:

- Ажил нь бүхэлдээ эсвэл ихэнхдээ Монгол Улсын Их Сургуулийн зэрэг горилохоор дэвшүүлсэн болно.
- Энэ ажлын аль нэг хэсгийг эсвэл бүхлээр нь ямар нэг их, дээд сургуулийн зэрэг горилохоор оруулж байгаагүй.
- Бусдын хийсэн ажлаас хуулбарлаагүй, ашигласан бол ишлэл, зүүлт хийсэн.
- Ажлыг би өөрөө (хамтарч) хийсэн ба миний хийсэн ажил, үзүүлсэн дэмжлэгийг дипломын ажилд тодорхой тусгасан.
- Ажилд тусалсан бүх эх сурвалжид талархаж байна.

Гарын үсэг: _____

Огноо: _____

ГАРЧИГ

ЗУРГИЙН ЖАГСААЛТ

ХҮСНЭГТИЙН ЖАГСААЛТ

Кодын жагсаалт

УДИРТГАЛ

Сүүлийн жилүүдэд хиймэл оюуны салбар дахь технологийн хурдацтай хөгжил нь програм хангамж хөгжүүлэлтийн арга барилд үндсэндээ өөрчлөлт авчирсан. Тухайлбал, суурь загвар гарч ирэх нь аппликейшн хөгжүүлэлтийн өмнө тулгардаг саад бэрхшээлийг эрс багасгасан бөгөөд энэ нь хиймэл оюуны инженерчлэл гэсэн шинэ салбарыг бий болгоход хүргэжээ. Goldman Sachs-ийн судалгаагаар 2025 он гэхэд АНУ-д хиймэл оюуны хөрөнгө оруулалт 100 тэрбум ам.доллар, дэлхий даяар 200 тэрбум ам.долларт хүрнэ гэсэн таамаглал дэвшүүлжээ [2].

Хиймэл оюуны инженерчлэл гэдэг нь бэлтгэгдсэн суурь загвар дээр аппликейшн бүтээх үйл явц юм. Энэхүү чиг хандлагын ач холбогдол нь хиймэл оюуны аппликейшнуудын эрэлт нэмэгдэхийн зэрэгцээ, тэдгээрийг бүтээх саад бэрхшээл багассан явдал юм. Машин сургалт дээр загвар бэлтгэхэд өндөр мэргэжлийн ур чадвар болон асар их өгөгдөл шаардлагатай байсан бол одоо бэлэн загварыг ашиглан аппликейшн хөгжүүлэх боломжтой болсон.

Энэхүү хөгжил нь микросервис архитектур дээр тулгуурласан програм хангамжийн хөгжүүлэлтэд онцгой боломжуудыг нээж өгч байна. Микросервис архитектур нь том системийг жижиг, бие даасан сервисүүдэд хуваах замаар уян хатан, өргөжүүлэх боломжтой, найдвартай системүүд бий болгодог. Гэвч эдгээр микросервис хоорондын харилцаа холбоо, өгөгдлийн урсгалыг оновчтой удирдах, хэрэглэгчийн хүсэлтийг олон сервисүүдийн хамтын ажиллагаагаар шийдэх нь нарийн төвөгтэй асуудал байсаар ирсэн.

Хиймэл оюун агентууд (AI agents) нь энэхүү асуудалд шинэлэг шийдэл санал болж байна. Агент гэдэг нь өөрийн орчныг мэдрэх, түүн дээр үйлдэл хийх чадвартай систем юм [1]. Хиймэл оюун агентууд нь том хэлний загварын хүчийг ашиглан даалгавруудыг ойлгож, төлөвлөгөө гаргаж, олон алхам бүхий үйл ажиллагааг гүйцэтгэх чадвартай. Эдгээр агентуудыг микросервис архитектурт нэвтрүүлэх нь системийн ухаалаг орчуулагч, өгөгдөл боловсруулагч, ажлын урсгалын удирдагч зэрэг үүргийг гүйцэтгэх боломжийг олгоно.

Энэхүү судалгааны ажил нь хиймэл оюун агентууд болон микросервис архитектурын уялдааг судалж, практик шийдлүүдийг санал болгохыг зорьж байна. Ялангуяа, суурь загварын онол, агентуудын төлөвлөлт, хайлтаар нэмэгдүүлсэн үүсгэлт (Retrieval-Augmented Generation) болон эдгээрийг микросервис архитектурт хэрхэн нэгтгэх талаар авч үзэх юм.

Нэр томьёоны тайлбар

AI Engineering Хиймэл оюуны инженерчлэл

Foundation Model Суурь загвар

Language Model Хэлний загвар

Masked Language Model Далдлагдсан хэлний загвар

Large Language Model Том хэлний загвар

Labeled data Тэмдэглэгдсэн өгөгдөл

Retrieval-Augmented Generation Хайлтаар нэмэгдүүлсэн үүсгэлт

Retrieval algorithm Хайлтын алгоритм

Prompt Engineering Зааврын инженерчлэл

Machine Learning Машин сургалт

Deep Learning Гүн сургалт

Microservices Architecture Микросервис архитектур

Event-Driven Architecture Үйл явдлаар удирдагдах архитектур

Message Queue Мессежийн дараалал

Vector Database Векторын өгөгдлийн сан

Embedding Векторчилсон хэсгийн төлөөлөл

Token Токен (хэлний загвард тэмдэгт мөрийн таслалын нэгж)

Inference Гаргалгаа (загвараас үр дүн гаргах үйл явц)

Fine-tuning Нарийвчилсан сургалт

Pre-training Урьдчилсан сургалт

Supervised Learning Удирдлагатай сургалт

Self-supervised Learning Өөрийгөө удирдсан сургалт

Reinforcement Learning Бэхжүүлсэн сургалт

Temperature Температур (загварын бүтээлч чанарын параметр)

Hallucination Төөрөгдөл (загварын буруу мэдээлэл үүсгэх үзэгдэл)

Context Window Контекстийн цонх

Agent Агент (бие даан үйлдэл хийх систем)

Tool Хэрэглүүр (хиймэл оюуны ашиглах хэрэглүүрүүд, ихэвчлэн API хэлбэрээр дамждаг.)

Orchestrator Зохион байгуулагч

Consumer Хэрэглэгч (мессеж хүлээн авагч)

Producer Үйлдвэрлэгч (мессеж илгээгч)

Topic Сэдэв (Kafka-гийн мессежийн ангилал)

Partition Хэсэглэл

Stream Processing Урсгал боловсруулалт

Latency Хоцрогдол

Throughput Дамжуулалт

Scalability Өргөжих чадвар

Resilience Уян хатан чанар байдал

Coupling Хамаарал

Decoupling Салангид байдал

Synchronous communication Синхрон холбоо

Asynchronous communication Асинхрон холбоо

Товчилсон үгс

LLM Large Language Model — Том хэлний загвар

RAG Retrieval-Augmented Generation — Хайлтаар нэмэгдүүлсэн үүсгэлт

API Application Programming Interface — Програмчлалын интерфэйс

REST Representational State Transfer — Төлөв байдлын төлөөлөлийн дамжуулалт

HTTP Hypertext Transfer Protocol — Гипертекст дамжуулалтын протокол

JSON JavaScript Object Notation — JavaScript объектын тэмдэглэгээ

SQL Structured Query Language — Бүтэцлэгдсэн асуулгын хэл

BERT Bidirectional Encentations foder Represrom Transformers — Transformer архитектурт суурилсан хэлний загвар

GPT Generative Pre-trained Transformer — Урьдчилан сурсан үүсгэгч Transformer

NLP Natural Language Processing — Байгалийн хэлний боловсруулалт

ML Machine Learning — Машин сургалт

MLOps Machine Learning Operations — Машин сургалтын үйл ажиллагааны удирдлага

EDA Event-Driven Architecture — Үйл явдлаар удирдагдах архитектур

SFT Supervised Fine-tuning — Удирдлагатай нарийвчилсан сургалт

RLHF Reinforcement Learning from Human Feedback — Хүний санал хүсэлтээр бэхжүүлсэн сургалт

DPO Direct Preference Optimization — Шууд сонголтын оновчлол

TF-IDF Term Frequency-Inverse Document Frequency — Нэр томьёоны давтамж ба баримтын урвуу давтамж

k-NN k-Nearest Neighbors — k хамгийн ойр хөршүүд

ANN Approximate Nearest Neighbors — Ойролцоо хамгийн ойр хөршүүд

FAISS Facebook AI Similarity Search — Facebook-ийн хиймэл оюунт семантик хайлтын сан

gRPC Google Remote Procedure Call — Google-ийн алсын процедур дуудлага

SOAP Simple Object Access Protocol — Объект хандалтын энгийн протокол

MSE, МХБ Mongolian Stock Exchange — Монголын Хөрөнгийн Бирж

CRUD Create, Read, Update, Delete — Үүсгэх, унших, шинэчлэх, устгах үйлдлүүд

JWT JSON Web Token — JSON форматаар илгээгддэг вэб токен

SSE Server-Sent Events — Серверээс илгээгдэх үзэгдлийн урсгал

VaR Value at Risk — Эрсдэлийн үнэлгээ

Техникийн нэр томъёо

Docker Контейнержуулалтын платформ; сервис бүрийг тусдаа орчинд ажиллуулах боломж олгодог.

Apache Kafka Салангид урсгалын платформ; өндөр дамжуулалттай мессежийн брокер.

Apache Flink Урсгал боловсруулалтын фреймворк; бодит цагийн өгөгдөл боловсруулах хэрэглүүр.

PostgreSQL Өгөгдлийн сангийн систем.

Redis Санах ойд суурилсан өгөгдлийн сан; кэш болон богино хугацааны өгөгдөл хадгалах зориулалттай.

Node.js JavaScript-ын ажиллах орчин; сервер талын хөгжүүлэлтэд ашиглагдана.

TypeScript JavaScript-ийн супер багц хэл.

Python Python Програмчлалын хэл; өгөгдөл боловсруулалт, AI хөгжүүлэлт зэрэгт ашиглагддаг.

Next.js React суурилсан веб фреймворк.

Express.js Node.js суурилсан веб фреймворк; RESTful API хөгжүүлэхэд өргөн ашиглагддаг.

Zookeeper Салангид зохицуулалтын сервис; Kafka зэрэг системийн мета өгөгдөл, кластерийн төлвийг удирдана.

Зорилго:

Энэхүү судалгааны ажлын гол зорилго нь хиймэл оюун агентуудыг микросервис архитектур хэлбэрээр нэвтрүүлэх боломжийг онол, практикийн хувьд судалж, үйл явдлаар удирдагдах архитектур (EDA) ашиглан уян хатан, өргөжих боломжтой системийн зохиомж гаргах юм. Уг зохиомжийн үр ашигтай байдлыг баталгаажуулахын тулд Монголын хөрөнгийн биржийн бодит өгөгдөлд суурилсан демо систем хөгжүүлж туршина.

Зорилт:

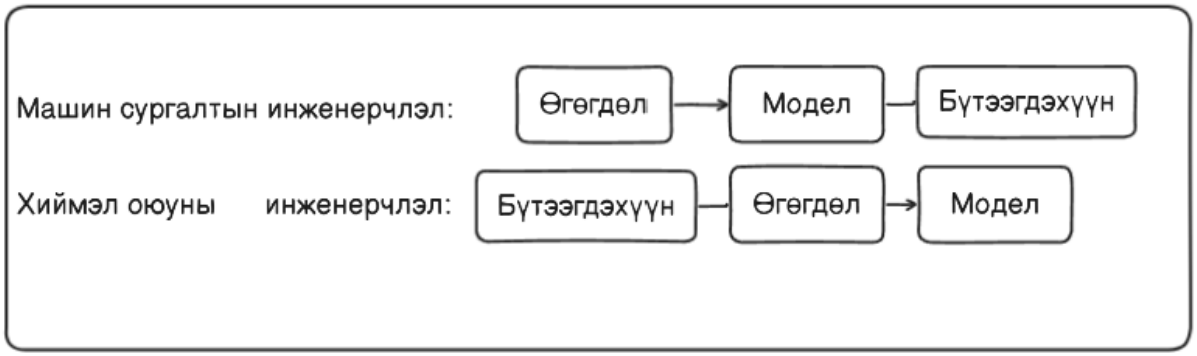
1. Хиймэл оюуны инженерчлэл, суурь загвар, RAG систем, агентуудын онолын үндсийг судлах
2. Микросервис архитектур болон үйл явдлаар удирдагдах архитектурын давуу талыг тодорхойлох
3. Apache Kafka, Apache Flink ашиглан агентуудыг тархмал микросервист нэгтгэсэн зохиомж гаргах
4. Монголын хөрөнгийн биржийн өгөгдөлд суурилсан демо систем хөгжүүлж, санал болгосон зохиомжийг туршиж үзэх

1. ХИЙМЭЛ ОЮУНЫ ИНЖЕНЕРЧЛЭЛ БА АГЕНТУУД

Энэхүү бүлэгт хиймэл оюуны инженерчлэлийн үндсэн ойлголтууд, суурь загварын хөгжил, промпт инженерчлэл, дараа нь RAG систем, агентуудын архитектурыг тайлбарлана. Мөн ижил төстэй системүүдийн судалгааг хийж, энэхүү судалгааны ажлын онцлог байр суурийг тодорхойлно.

1.1 Хиймэл оюуны инженерчлэл гэж юу вэ

Хиймэл оюуны инженерчлэл гэдэг нь бэлэн бэлтгэгдсэн суурь загвар дээр аппликейшн хөгжүүлэх үйл явцийг хэлнэ. Энэ нь уламжлалт машин сургалтын инженерчлэлээс ялгаатай байдаг [1]. Хэрэв уламжлалт машин сургалтын инженерчлэл нь загвар хөгжүүлэхэд чиглэсэн бол, хиймэл оюуны инженерчлэл нь бэлтгэгдсэн загварыг программ хангамжид интеграц хийхэд чиглэсэн байдаг. Энэ ч утгаараа фүллстэк хөгжүүлэгч заавал машин сургалтын



Зураг 1.1: Хиймэл оюуны инженерчлэл ба машин сургалтын инженерчлэл [1]

Зураг 1.1-д хиймэл оюуны инженерчлэл ба машин сургалтын инженерчлэлийн ялгааг дүрсэлсэн. Зураг 1.1-оос харахад хиймэл оюуны инженерчлэл нь бэлэн загварыг ашиглахад чиглэсэн бол машин сургалтын инженерчлэл нь загвар хөгжүүлэхэд чиглэсэн байна.

OpenAI, Anthropic зэрэг компаниудын гаргасан хүчирхэг суурь загварын хүртээмж нь гурван гол хүчин зүйлээс шалтгаалан хиймэл оюуны инженерчлэлийг хурдан өсч буй салбар болгожээ. [2] Эхнийх нь өндөр эрэлт юм. Компаниуд хиймэл оюуныг бусад бизнесээс ялгарах өрсөлдөөнт давуу тал болгон үзэж байна. FactSet-ийн судалгаагаар 2023 оны хоёрдугаар улиралд S&P 500 компаниудын гуравны нэг нь өөрсдийн санхүүгийн тайланд хиймэл оюуныг дурдсан тоо нь өмнөх оноос гурав дахин их байна. Хоёр дахь нь технологийн хөгжилтэй холбоотой. Өмнө нь хиймэл оюун систем бүтээхэд өндөр мэргэжлийн ур чадвар, их хэмжээний өгөгдөл, тооцооллын нөөц шаардлагатай байсан бол одоо суурь загвар ашиглан хэрэглээнд нэвтрүүлэх нь илүү боломжтой болов. Улмаас хиймэл оюуны инженерүүд программ хангамжийг хүртээмжтэй загварууд ашигласнаар өндөр түвшний математикийн мэдлэг өндөр түвшинд байх шаардлагагүй болов. Гурав дахь нь том боломж юм. Хиймэл оюун технологи нь ажил хэргийг автоматжуулах, шинэ бүтээгдэхүүнүүдийг бий болгох зэрэг асар том боломжуудыг санал болгож байна. Энэхүү хандлагын нотолгоо болох хиймэл оюунт аппликейшний нээлттэй эхийн кодууд (Auto-GPT, Stable Diffusion WebUI, LangChain, Ollama) нь GitHub дээр Bitcoin-оос ч илүү од цуглуулсан нь энэхүү салбарын хурдацтай өсөлтийг тод илэрхийлж байна. [1].

1.2 Суурь загварын хөгжил

Хэл загваруудээс том хэлний загвар болон суурь загвар руу хөгжих үйл явц нь хэдэн арван жилийн технологийн дэвшлийн үр дүн юм. Энэхүү хэсэгт гол түлхүүр үйл явдлыг тайлбарлах болно. [1]

1.2.1 Хэл загварын үндэс

Хэл загвар гэдэг нь нэг буюу олон хэлүүдийг статистик өгөгдөл рүү кодлодог загвар юм. Энэхүү мэдээлэл нь өгөгдсөн контекстэд уг үг гарах магадлалыг илэрхийлдэг. Жишээлбэл, ”Миний дуртай өнгө бол __” гэсэн контекст өгөхөд монгол хэлээр кодолсон хэл загвар нь ”машин” биш, харин ”цэнхэр” гэсэн үгийг таамаглах ёстой. [1]

Анхны текстийг токен болгон хуваах үйл явцыг токенжуулалт гэнэ. GPT-4 суурь загварын хувьд дунджаар нэг токен нь үгийн ойролцоогоор 75%-ийн уртад тохирно. Тиймээс 100 токен нь ойролцоогоор 75 үг юм.

Хэл загвард хоёр үндсэн төрөл байдаг. Эхний төрөл нь далдлагдсан хэлний загварууд бөгөөд эдгээр нь өгүүлбэр доторх далдлагдсан үгсийг таамаглах замаар сурдаг. BERT нь энэ төрлийн алдартай жишээ юм. Хоёр дахь төрөл нь авторегрессив хэл загварууд бөгөөд өмнөх токенуудад үндэслэн дараагийн токенийг таамаглах замаар сурдаг. Одоогийн Chat-GPT, Claude зэрэг өргөн ашиглагдаж буй системүүд нь энэ ангилалд хамаарагддаг.

1.2.2 Өөрийгөө удирдсан сургалт (Self-Supervised Learning)

Хэл загварын хамгийн чухал давуу тал нь өөрийгөө удирдсан сургалтыг ашиглах чадвар юм [1]. Өөрийгөө удирдсан сургалт нь удирдлагатай сургалтаас ялгаатай байдаг. Удирдлагатай сургалт нь тэмдэглэгдсэн өгөгдөл шаарддаг бөгөөд энэ үйл явц нь цаг хугацаа их зарцуулдаг. [1]

Энэ нь хэл загварыг номнуудаас, блог нийтлэлээс, өгүүллүүд, Reddit-ийн сэтгэгдэл зэргээс ашиглан сургаж болно. Энэ нь асар их сургалтын өгөгдөл бүрдүүлэх боломжийг олгож, хэл загварыг том хэлний загвар буюу LLM болтол өргөжүүлэх боломжтой болгосон.

1.2.3 Том хэлний загвараас суурь загвар руу

2017 онд Transformer архитектур гарч ирснээр хэл загварын чадамж харьцангуй өндөр нэмэгдсэн. Attention механизм нь загваруудад өгөгдлийн хамаарлыг илүү сайн ойлгох боломжийг олгосон. [1]

Том хэлний загварууд нь хэл загварын томорсон хувилбар бөгөөд тэрбум тооны параметр агуулдаг. Параметр гэдэг нь сургалтын явцад загварын сурч авдаг утга юм. Жишээлбэл, GPT-3 нь 175 тэрбум параметртэй, харин GPT-4 нь 1.2 их наяд параметртэй байдаг.

Суурь загварууд нь LLM-ээс цааш өргөжсөн ойлголт юм. Эдгээр нь зөвхөн текст биш, зураг, аудио, видео зэрэг олон төрлийн өгөгдөл боловсруулж чаддаг том мульти

модал загварууд юм. Суурь загварын гол онцлог нь тодорхой үүрэгтэй загвараас цаашлаад ерөнхий зориулалтын загвар руу шилжсэн юм.

1.3 Хиймэл оюуны агент ба бизнесийн үйл ажиллагаа

Энтерпрайзийн хувьд хиймэл оюун нь дахин давтагддаг нэхэмжлэл үүсгэх, харилцагчийн асуултад хариулах, өгөгдөл бүртгэх зэрэг үйл ажиллагааг автоматжуулах боломж гаргаж байдаг. Нэгэн сонирхолтой хиймэл оюуны нэвтрүүлэлтийн хэлбэр нь өгөгдлийг дахин сайжруулах арга зам юм. Энэ нь өөрийнхөө өгөгдөлд тэмдэглэгээ хийгээд, дараа нь энэ тэмдэглэгээний үр дүнгээс хамаарч жинхэнэ хүний тусламжтайгаар алдааг багасгахын тулд олон дахин уг тэмдэглэгээнүүдийг сайжруулах юм. Энтерпрайзуудад хамгийн алдартай хиймэл оюуны хэрэглээ нь харилцагчийн туслах бот юм. Туслах бот нь хүнээс хурдан хариулснаар харилцагчийн туршлагыг сайжруулж, бас бизнесийн хувьд зардал хэмнэх боломжийг бүрдүүлдэг. [1] [2]

Хиймэл оюунд гадна орчинтой хандах хэрэглүүр өгөх нь маш олон боломжийг нээж өгдөг. Ресторанд цаг захиалахад сул цаг харах, захиалга өгөх зэрэг үйлдлийг хэрэглүүрүүд хийх боломжтой ба хиймэл оюунд уг хэрэглүүр өгснөөр харилцагчийн өмнөөс уг үйлдлийг автоматаар хийж болох юм. Уг үйлдлүүдийг зохион байгуулж, хэрэглүүр ашигладаг программ хангамжийг хиймэл оюун агентууд гэнэ.

1.4 Суурь загварын сургалт

Суурь загварыг бэлтгэх нь хоёр үндсэн үе шаттай:

1.4.1 Урьдчилан сургалт

Урьдчилан сургалт нь өөрийгөө удирдсан сургалт ашиглан их хэмжээний өгөгдөл дээр загварыг сургах үйл явц юм. Энэ үе шатанд загвар нь хэл, ерөнхий мэдлэг, дүрэм, баримт бичгүүдээс суралцдаг. 2022 онд сургалтын дата олохын тулд нэгэн ашгийн бус байгууллага 2-3 тэрбум веб хуудсуудыг автоматаар авч сургасан байна. Гэвч худал хуурмаг мэдээлэл, хүнд сурталтай мэдээлэл их байдаг тул хьюристик филтер хийдэг.

Жишээ нь реддит платформд 5-аас олон эерэг хариу үйлдэлтэй бол уг өгөгдлийг авах юм. [1]

Суурь загвар нь олон төрөлтэй байна. Нэгт, тодорхой зорилготой агентууд нь домейнд л хамаарагдах өгөгдлийг ашиглаж тооцоолол хийдэг. Үүнд эм эмчилгээний жорыг гаргах, ДНХ, хавдрын, уурагны симуляц явах зэрэг үйлдлүүдтэй байна. Хоёрт, ерөнхий зориулалттай хиймэл оюуны загвар байна. Сургагдсан өгөгдлийн талаас дээш хувийг технологи, бизнес, үйлдвэр, мэдээ, урлаг уран сайхны өгөгдлүүд эзлэх ба үлдсэн хувийг бусад бага бага хувьтай гэр, аялал зэрэг секторууд эзлэж байна.

Олон улсын хэлүүд суурь загвар дээр өөр өөр ажиллах зарчимтай байна. Сургалтан дээр суурь загварын ойролцоогоор тал хувийг англи хэл эзэлдэг бол орос, герман хэл 10 хувийг эзлэж байна. GPT-4 суурь загварын хувьд ашиглах токений хэмжээ ба төөрөгдөл хамгийн бага байх ба, Бирм хэл англи хэлээс 70 дахин их токен ашиглаж, төөрөгдөл хамгийн өндөр байна. Шалтгаан нь уг хэлний соёлийн бүтэц юм. Жишээлбэл зарим хэлд эзэн бие ашигладаггүй учир хэл хөрвөхөд оновчтой байх магадлал бага юм.

Иймээс суурь загварын хэлний хязгаарлалтыг давахын тулд өөрсдийн хэл дээр суурь загвар хөгжүүлж байна. Жишээлбэл, хятадын ”Llama-Chinese”, франц ийн ”Croissant-LLM”, Виетнамын ”PhoGPT” гэх зэрэг. Монгол улсын хувьд ”Чимэгэ систем” нь монгол хэл дээр суурь загвар хөгжүүлж байгаа.

Урьдчилан сургагдсан үе шатд хэрэглэгчдийн хүсэлтэд нийцсэн хариулт өгөхөд сайн биш байдаг. Учир нь харилцан яриа өрнүүлэх гэхээс илүүтэйгээр зөвхөн өгүүлбэрийн гүйцээлт рүү тулгуурлан сургагдсан байдаг. Иймээс дараах сургалт, sampling техникүүд, нарийвчилсан сургалтууд шаардалагатай байдаг.

1.4.2 Дараах сургалт

Урьдчилан сургасан загварыг хэрэглэгчдийн хүсэлтэд тохируулахын тулд дараах сургалт хийдэг. Энэ нь хоёр үе шаттай. Эхний үе шат нь удирдлагатай нарийвчилсан сургалт юм. Энэ үе шатанд өндөр чанартай зааварчилгааны өгөгдөл дээр загварыг нарийвчлан сургаж, зөвхөн өгүүлбэрийн гүйцээлт биш харин харилцан ярианы горимд оновчтой болгоно. Хоёр дахь үе шат нь сонголтын нарийвчилсан сургалт юм. Энэ үе

шатанд загварыг хүний сонголттой нийцсэн хариулт өгөхийн тулд цаашид нарийвчлан сургана. Үүнд хүний санал хүсэлтээр бэхжүүлсэн сургалт, хиймэл оюуны санал хүсэлтээр бэхжүүлсэн сургалт зэрэг аргууд ордог. [1]

1.4.3 Sampling стратегиуд

Суурь загвараас гарах гаралт нь адилхан асуулт өгсөн ч хариулт бүр тогтмол биш, ялгаатай байх нь магадлалын шинж чанарыг илтгэдэг. Уг суурь загварын гаралтыг sampling процессоор хийх ба sampling нь хиймэл оюуны гаралтын магадлалд шууд нөлөөлдөг [1]. Үүний чухал параметр болох температур нь загварын бүтээлч чанарыг удирдах гол параметр юм. Температур өндөр байх тусам загвар илүү бүтээлч, гэнэтийн хариулт өгдөг бол температур бага байвал хамгийн илэрхий хариултыг сонгож, илүү баттай хариулт өгдөг. Top-k нь хамгийн их магадлалтай k ширхэг токеноос сонгодог арга юм. Үүнийг өөрчилснөөр хариултын олон янзтай байх байдлыг удирдаж болно. Энэхүү арга нь таамаглагдашгүй содон урт уран зохиол бичихэд чухал. Түүнчлэн nucleus sampling буюу Top-p арга байдаг. Энэ нь нийлбэр магадлал нь p-д хүрэх хамгийн бага токенуудын багцаас сонгодог. Энэ нь тийм, үгүй эсвэл урт хариулт, богино хариултын загварыг тодорхойлдог. Иймээс хэрэглээнээс хамаарч samping стратеги сонгох нь чухал. Жишээлбэл тийм, үгүй сонголттой асуулт хариулт гаргах, урт тэмдэгт мөртэй хариулт гаргах гэх мэт байж болно.

1.4.4 Загварын үр дүнг хэмжих

Суурь загварыг үнэлэх нь эрсдэлийг бууруулах, цаашлаад боломжуудыг илрүүлэх тал дээр чухал ач холбогдолтой. Үнэлгээ нь загвар сонгох, үр дүнг хэмжих, аппликейшн ашиглалтад бэлэн эсэхийг тодорхойлох, асуудал болон боломжуудыг илрүүлэх зэрэгт шаардлагатай. [1]

Сүүлийн жилүүдэд бага параметртэй загвар нь өмнөх үеийн их параметртэй загвараас илүү чадалтай байна. Жишээлбэл, 2024 оны Llama 3-8B загвар нь 2023 оны Llama 2-70B загвараас ч илүү сайн үр дүнг MMLU benchmark дээр харуулжээ. Энэ нь зөвхөн загварын

хэмжээ биш, сургалтын аргууд болон өгөгдлийн чанар хамгийн чухал болохыг харуулж байна.

Үнэлгээний хувьд гурван гол асуудал тулгардаг. Нэг дүгээрт, суурь загваруудыг зөвхөн гаралтуудын өгөгдлөөс дүгнэж үнэлэхэд хэцүү байдаг. Үүнийг хар хайрцаг гэх ба дотоод ажиллаж байгаа үйл явц биш зөвхөн эцсийн гаралт нь л мэдэгдэж байдаг. Хоёр дугаарт, загвар нь ижил эсвэл бага зэрэг өөр асуулт асуухад маш өөр хариулт өгч болох тогтворгүй байдал байдаг. Үүнээс хиймэл оюуны суурь загварын хариулт нь магадлалаас үүсдгийг ажиглаж болно. Гуравдугаарт, загвар нь баримт дээр үндэслээгүй буруу хариулт буюу төөрөгдөл үүсгэж болдог.

1.5 Промпт инженерчлэл

Промпт инженерчлэл гэдэг нь загвараас хүссэн үр дүнг гаргуулахын тулд промпт заавар бичих үйл явц юм [7]. Энэ нь загварын жинг өөрчлөхгүйгээр зан үйлийг удирдах хамгийн хялбар бөгөөд түгээмэл загварын дасан зохицох арга юм.

1.5.1 Промпт бичих шилдэг арга барил

Промпт зааврууд хэрэглээнд тохирсон стратегиудын дагуу бичих нь илүү сайн үр дүн өгдөг. [7] OpenAI-ийн санал болгож буй промпт бичих шилдэг арга барил нь эхлээд юу хийлгэхээ хоёрдмол утгагүй байдлаар тодорхой тайлбарлах хэрэгтэй. Дараа нь загвараар тодорхой дүрд тоглуулж болно. Жишээ нь ”Та том компанид 20 жил ажилласан туршлагатай программист. Кодыг шалгаад сайжруулж өг” гэх мэт. Anthropic-ийн зөвлөмжөөр промптод 500 хуудас бүхий номын урттай тэмдэгт мөр багтаж чадах тул промптод урт жишээ өгснөөр хариултын формат болон хариултын хоёрдмол утгыг багасгадаг. Цар хүрээ, агуулгыг мэдээлүүлснээр төөрөгдлийг багасгадаг. Хэрэв загвар шаардлагатай мэдээллээр хангагдаагүй бол өөрийн дотоод мэдлэгтээ найдах бөгөөд энэ нь найдваргүй байж болдог. Түүнчлэн нарийн төвөгтэй даалгаврууд ыг хялбар дэд даалгаврууд ад хувааж өгөх нь үр дүнтэйгээр бага токен ашиглах боломжийг олгоно.

1.5.2 Зааврын инженерчлэлийн хамгаалалт

Аппликейшн олон нийтэд ашиглагдах үе шатанд ормогц довтолгооноос хамгаалах шаардлагатай болдог. Тэдгээрийн нэг нь загварын зөвшөөрөөгүй үйлдэл хийлгэх оролдлого юм. Нөгөө нь загварын сургалтын өгөгдөл эсвэл контекстын мэдээллийг задруулах оролдлого юм. [1]

Иймээс хиймэл оюуны агент эсвэл загвар угсарч буй тохиолдолд ролт болон гаралтыг үнэлж хамгаалах функц нэвтрүүлэх нь мэдээллийн аюулгүй байдлыг хангадаг.

1.6 Хайлтаар нэмэгдүүлсэн үүсгэлт (RAG)

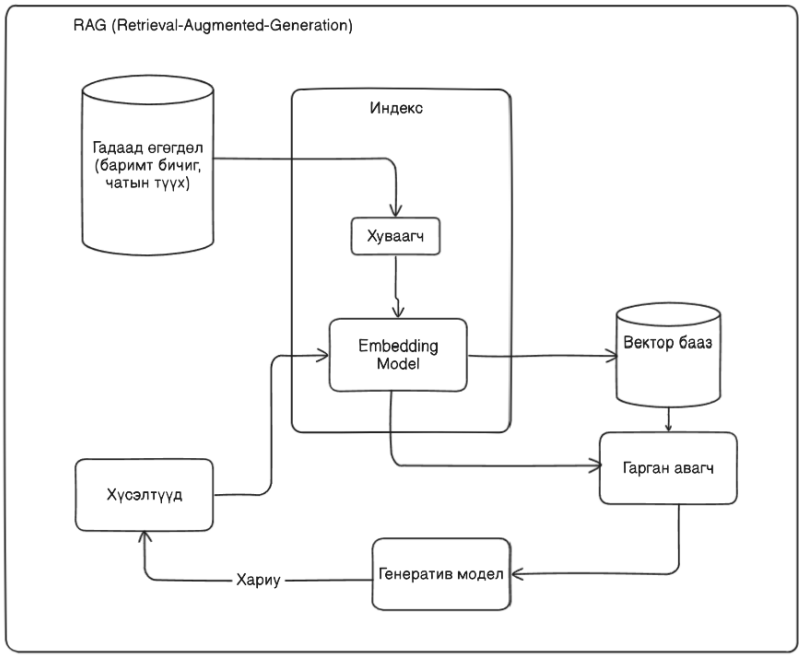
RAG буюу хайлтаар нэмэгдүүлсэн үүсгэлт нь загварын мэдлэгийг гадаад эх сурвалжаар өргөтгөх арга юм. Энэ нь загварын дотоод мэдлэг нь хангалтгүй, хуучирсан эсвэл алдаатай байх асуудлыг шийддэг. [1]

Хэдийгээр загварын контекстийн урт тогтмол нэмэгдэж байгаа ч RAG-ийн ач холбогдол алдагдахгүй байна. Зарим аппликейшнд өгөгдлийн хэмжээ байнга өсч байдаг. Иймээс RAG удааширж магадгүй тул үүнийг сайтар үнэлж, тасралтгүй хөгжүүлэлт хийснээр бодит хэрэглээнд үр нөлөө алдахгүй байх боломжийг бүрдүүлнэ. Урт контекстыг боловсруулж чаддаг гэдэг нь тэр контекстыг сайн ашигладаг гэсэн үг биш. Контекст урт байх тусам загвар буруу хэсэгт анхаарал хандуулах магадлал өсдөг. Түүнчлэн контекстын токен бүр нэмэлт өртөг, нэмэлт хоцрогдол авчирдаг. RAG нь асуулт бүрт зөвхөн хамгийн холбогдолтой мэдээллийг ашиглах боломжийг олгоно.

Anthropic-ийн зөвлөмжөөр хэрэв мэдээлэл нь 200,000 токеноос бага (ойролцоогоор 500 хуудас бүхий өгөгдөл) бол RAG ашиглалгүй бүх мэдлэгийг промпт зааварт оруулж болно гэжээ.

1.6.1 RAG системийн бүтэц

RAG систем нь хоёр гол бүрэлдэхүүнтэй. Хайгч нь асуултад хамгийн холбогдолтой баримтуудыг олж авдаг. Үүсгэгч нь олж авсан баримтуудыг асуултад ашиглан хариулт үүсгэдэг.



Зураг 1.2: RAG-ийн бүтэц

Зураг 1.2-аас харахад хайгч нь асуултад хамгийн их холбоотой баримтуудыг олж авдаг бол үүсгэгч нь олж авсан баримтуудыг ашиглан хариулт үүсгэдэг.

1.6.2 Хайлтын алгоритмууд

Хайлтаар олдсон өгөгдөл нь хэр оновчтой байх нь RAG-ийн хамгийн чухал хэсгүүдийн нэг [1]. Өгөгдлийг вектор эсвэл өгөгдлийн бааз руу оруулах хялбар ч үүнээс хайлт хийх нь харьцангуй хүнд байдаг. Хамгийн түгээмэл алдаа нь векторд хэсэгчилж хуваагдахад өгүүлбэрүүд утга зүй бусаар хуваагдаж, хайлт хийх боломжгүй болдог. Иймээс хайлтын алгоритмээ зөв сонгох нь маш чухал. Дараах үндсэн хайлтын аргууд байдаг [1].

Нэр томьёо суурилсан хайлт

Энэ арга нь түлхүүр үгээр баримт хайдаг. Энэ арга Google, Bing зэрэг хөтчийн хайлтын алгоритмд ашиглагдсаар ирсэн. Нэр томьёоны давтамж нь баримт доор нэр томьёо хэдэн удаа гарч байгааг хэмждэг бол баримтын урвуу давтамж нь нэр томьёо хэдэн баримтад

гарч байгааг үндэслэн түүний чухлыг хэмждэг. Түгээмэл шийдлүүд нь Elasticsearch, BM25 зэрэг байдаг. Эдгээр нь урвуу индекс ашигладаг.

Утга зүй дээр суурилсан хайлт

Утга зүйн хайлт гэж нэрлэгддэг энэ арга нь утга зүйн түвшинд холбоотой байдлаар тооцож ажилладаг. Баримт бүр хуваагдаж embedding загвар болгон хувиргагдаж, векторын өгөгдлийн санд хадгалагдана. Асуулт ирэх үед түүний embedding загвартай хамгийн ойр векторуудыг хайдаг.

Векторын хайлтын алгоритмууд нь олон янз байдаг. Хамгийн ойр хөршүүд нь энгийн арга боловч өгөгдөл их бол удаан байдаг. Ойролцоо хамгийн ойр хөршүүд нь хурдан боловч ойролцоогоор хайдаг. Иймээс өгөгдлийн ангилал, форматаас шалтгаалж өөр өөр хайлтын алгоритм ашигладаг. Locality-Sensitive Hashing нь ижил төстэй векторуудыг нэг bucket-д hash хийдэг. Hierarchical Navigable Small World нь олон давхаргат граф ашигладаг. Inverted File Index нь K-means clustering ашиглан векторуудыг бүлэглэдэг. Алдартай векторын өгөгдлийн сангууд нь FAISS, Milvus, Pinecone, Weaviate, Qdrant зэрэг байна.

1.6.3 RAG-ын үнэлгээ

RAG системийг үнэлэхэд олон метрикүүд ашигладаг. [1]. Context Precision нь олж авсан баримтуудын хэдэн хувь нь асуулттай холбоотой эсэхийг хэмжинэ. Context Recall нь асуулттай холбоотой бүх баримтуудын хэдэн хувийг олж авсан эсэхийг илэрхийлнэ. Эцсийн хариултын чанар нь хариултын ерөнхий чанарыг үнэлдэг.

1.6.4 RAG-ыг сайжруулах аргууд

RAG системийг сайжруулах олон арга байдаг. Баримтуудыг хэрхэн хэсэглэж салгах нь чухал [1]. Тогтмол уртаар хэсэглэх, өгүүлбэр догол мөрөөр хэсэглэх, утга зүйгээр хэсэглэх зэрэг олон ялгаатай арга байдаг. Нэр томьёо болон утга зүйн хайлтыг хослуулсан арга ашиглаж болно. Асуултыг дахин найруулж илүү сайн хайлт хийж болно. Хэсэг бүрийг metadata, түлхүүр үг, холбогдох асуултуудаар баяжуулах нь хайлтын чанарыг

сайжруулдаг. Аливаа ажиллаж байгаа RAG-аас ялгаатай гаралтуудыг хадгалж үнэлвэл тохирсон хайлтын аргыг сонгоход ойлгомжтой болдог.

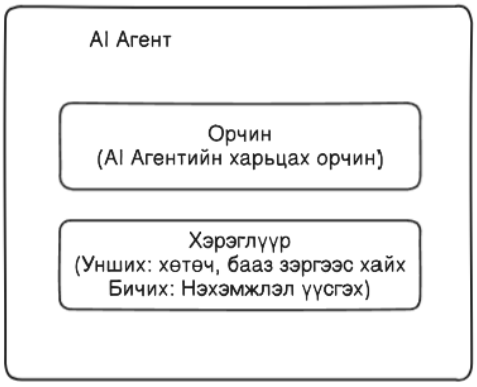
1.7 Хиймэл оюуны агентууд

1.7.1 Агент

Агент гэдэг нь өөрийн орчныг мэдрэх, түүн дээр үйлдэл хийх чадвартай систем юм. [1]. Хиймэл оюунаар дэмжигдсэн агентууд нь суурь загварын хүч чадлаар дамжуулан бидний туслах, хамтран ажиллагч, багш байж чадна. Агент нь вебсайт бүтээх, өгөгдөл цуглуулах, аялал төлөвлөх, зах зээлийн судалгаа хийх, харилцагчийн данс удирдах, өгөгдөл оруулалтыг автоматжуулах зэрэг олон ажил хэрэгт тусалж чадна.

1.7.2 Агентын бүрэлдэхүүн хэсгүүд

Хиймэл оюун агентыг тодорхойлдог хоёр гол зүйл байдаг. [1]. Орчин нь агент ажиллах орчин бөгөөд түүний хэрэглээний тохиолдлоор тодорхойлогдоно. Жишээ нь интернэт, гал тогоо, хөдөлгүүрт хэрэгсэл зэрэг байж болно. Агентын хийж чадах үйлдлүүд нь түүний хандах боломжтой хэрэглүүрүүдээр өргөжинө.



Зураг 1.3: Агентын бүрэлдэхүүн хэсгүүд

Зураг 1.3-т агентын бүрэлдэхүүн хэсгүүдийг дүрсэлсэн. Зураг 1.3-аас харахад агент нь орчин, үйлдэл, даалгавар гэсэн гурван гол бүрэлдэхүүнтэй.

1.7.3 Хэрэглүүрүүд

Гадаад хэрэглүүр байхгүй бол агентын чадавхи маш хязгаарлагдмал байх болно. Хэрэглүүр нь агентыг илүү чадварлаг болгодог. Цаашлаад уян хатан шийдвэр гаргалт, найдвартай гүйцэтгэлийн хоорондох хоосон зайг нөхдөг. Хэрэглүүрийг гурван ангилалд хуваах боломжтой.

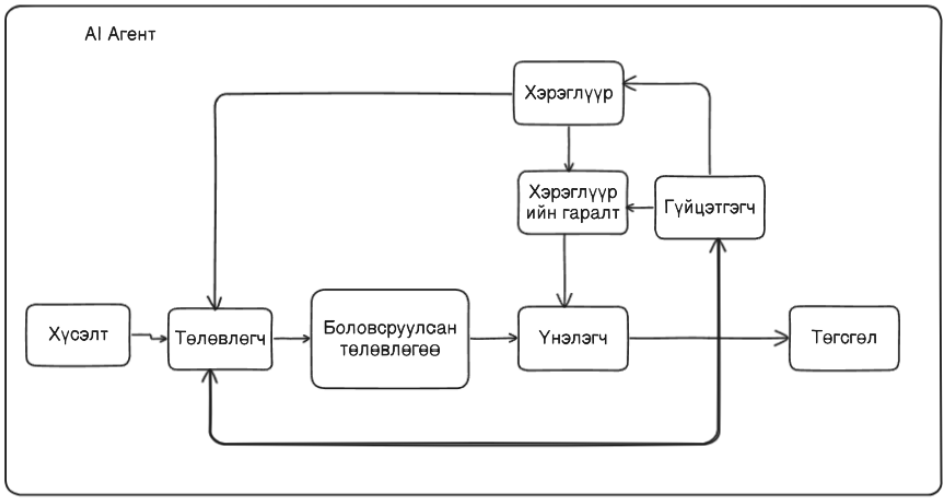
Мэдлэг нэмэгдүүлэх хэрэглүүрүүд нь хайлтаар нэмэгдүүлсэн үүсгэлт систем буюу RAG-ын бүрэлдэхүүн хэсгүүд юм. Үүнд текст хайгч, зураг хайгч, SQL гүйцэтгэгч, интернэт хайлтын програмчлалын интерфэйс, дотоод хайлтын системүүд зэрэг багтана.

Чадавх өргөтгөх хэрэглүүрүүд нь агентын үндсэн чадварыг өргөжүүлдэг. Хиймэл оюун загварууд математикт сул байдаг тул агентд бэлэн тооны машины програмчлалын интерфэйс өгснөөр тооцоог оновчтой, хурдан, токены бага зарцуулалтаар гүйцэтгэдэг. Хэрэглэгчийн код уншигч нь код бичиж, ажиллуулах, үр дүн гаргах чадвартай. Энэ нь кодчилолын туслах, өгөгдөл шинжлэгч, судалгааны туслах боломжийг олгоно.

Бичих үйлдлийн хэрэглүүрүүд нь зөвхөн унших биш, өөрчлөлт оруулах хэрэглүүрүүд юм. Үүнд өгөгдлийн санд өгөгдөл нэмэх, засварлах, устгах, мэйл илгээх, банкны шилжүүлэг хийх, календарт тэмдэглэл нэмэх зэрэг үйлдлүүд багтана. Бичих үйлдэл нь өндөр эрсдэлтэй байдаг. Иймээс хортой промптын довтолгооноос болгоомжлох хэрэгтэй.

1.7.4 Хиймэл оюун агентийн төлөвлөлт

Хиймэл оюун агентийн төлөвлөлт нь агентын гол үүрэг бөгөөд олон үе шаттай. Зураг 1.4-т эхний үе шат нь төлөвлөгөө үүсгэх юм. Төлөвлөгч нь аливаа даалгаврын дагуу аливаа процессийг гүйцэтгэх төлөвлөгөө боловсруулна. Хоёр дахь үе шат нь эргэцүүлэн бодох ба алдаа засах юм. Үүсгэсэн төлөвлөгөөг үнэлэх бөгөөд муу байвал шинэ төлөвлөгөө гаргана. Гурав дахь үе шат нь гүйцэтгэл юм. Төлөвлөгөөнд заасан үйлдлүүдийг хийнэ. Эцсийн үе шат нь үр дүнг үнэлэх явдал юм. Үйлдлийн үр дүнг хүлээн авсны дараа зорилго биелсэн эсэхийг тодорхойлж, алдааг засна.



Зураг 1.4: Агентын төлөвлөлт

1.7.5 Суурь загварууд төлөвлөгч болж чадах уу

Зарим судлаачид том хэлний загвар нь төлөвлөгч болж чадахгүй гэж үздэг. [16]. Учир нь төлөвлөлт нь үндсэндээ хайлтын асуудал бөгөөд авторегрессив буюу магадлалт суурилсан загвар нь зөвхөн үргэлж ялгаатай, буруу хариулт өгдөг. Гэвч бодит байдал дээр загвар өөр өөрийнхөө үүсгэсэн төлөвлөгөөг үнэлээд дахин эхэлж чаддаг тул сургалт сайн хийснээр сайн төлөвлөгөө гарч болно.

Төлөвлөлтийг сайжруулах олон арга байдаг. Илүү сайн системийн промпт заавар бичиж, жишээ олноор өгөх нь чухал. Хэрэглүүрүүдын тайлбарыг илүү сайн бичих хэрэгтэй. Функцүүдийг хялбарчлах, задлах нь төлөвлөлтийг хөнгөвчилдөг. Илүү хүчтэй загвар ашиглах нь илүү сайн төлөвлөгөө гаргах боломжийг олгоно. Төлөвлөлтөд зориулж загварыг нарийвчлан сургах нь бас үр дүнтэй. Зарим судалгаагаар оновчтой хэрэглүүр өгөх нь нарийвчилсан сургалт хийснээс илүү үр дүнтэй бас хямд гэж үзжээ [1].

1.7.6 Эргэцүүлэн бодох ба алдаа засах

Хамгийн сайн төлөвлөгөө байнга үнэлэгдэж, тохируулагдах шаардлагатай. Эргэцүүлэн бодох нь агентын амжилтад чухал үүрэг гүйцэтгэнэ. Үүнийг хоёр аргаар хийж болно. Эхний арга нь өөрийгөө шүүмжлэх арга юм. Ижил загвар өөртэйгөө ярилцаж алдааг

илрүүлдэг. Хоёр дахь арга нь тусдаа үнэлэгч ашиглах явдал юм. Тусдаа загвар эсвэл функц үр дүнд оноо өгдөг.

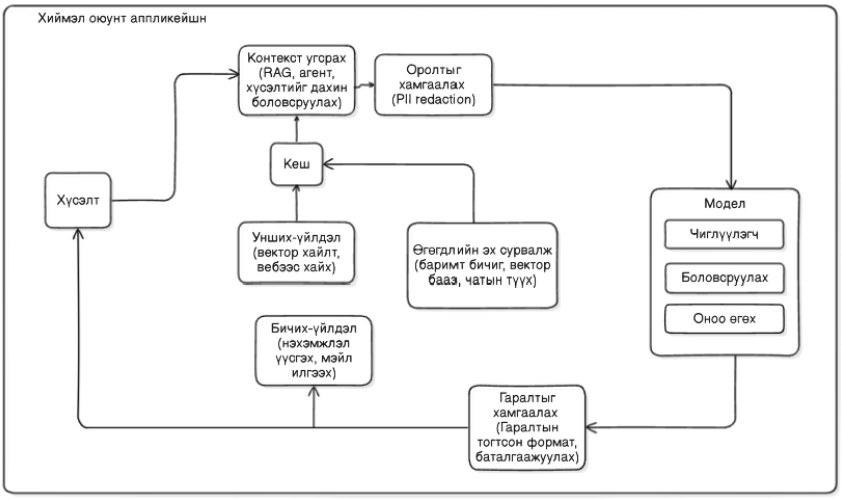
1.7.7 Агентын санах ой

Хиймэл оюун загвар нь гурван санах ойн механизмтай [1]. Дотоод мэдлэг нь загварын өөрийн дотоод мэдээлэл юм. Энэ нь сургалтын өгөгдлөөс олж авсан мэдлэг бөгөөд загварыг шинэчлэхгүй л бол өөрчлөгдөхгүй. Богино хугацааны санах ой нь загварын контекст юм. Өмнөх мессежүүд контекстэд нэмэгдэж болно. Даалгавар дууссаны дараа устдаг. Хурдан боловч устдаг шинжээр хязгаарлагдмал. Урт хугацааны санах ой нь гадаад өгөгдлийн эх сурвалж буюу хайлтаар нэмэгдүүлсэн үүсгэлт (RAG) юм.

1.7.8 Яагаад олон агентын зохиомж хэрэгтэй вэ?

Агент нь тодорхой даалгаврын хүрээнд л процесс гүйцэтгэх чадвартай байдаг. Жинхэнэ бизнесийн үйл ажиллагаа нь олон дэд процессд хуваагдаж болдог шиг агентийн хэрэглээ мөн адил олон хуваагдаж болдог. Тиймээс олон агентийн зохиомж нь ялгаатай орчинд нарийн төвөгтэй асуудлыг шийдэх, дасан зохицох, үр дүнтэй хамтран ажиллах боломжийг олгодог. Жишээлбэл байгууллага өөрсдийн сургасан үнэтэй загвартай байлаа гэж үзэхэд уг загварыг зөвхөн хэрэгтэй процессд нь ажиллуулаад, харин хямдхан GPT-2 зэрэг загвараар чиглүүлэгч, ангилагч зэрэг хямд процессд үлдээж болно. Иймээс олон агент зохиомжтой байх нь тооцооллын зардлыг бууруулж болох ба практикт тохиромжтой арга зам юм.

1.8 Жишээ хиймэл оюунт программ хангамжийн зохиомж



Зураг 1.5: Хиймэл оюунт аппликейшн

Зураг 1.5-т олон агентийн архитектурын зохиомжийг дүрсэлнэ. Программ хангамж хөгжүүлэхэд бэлэн суурь загвар эсвэл өөрсдөө байршуулсан загварыг ашиглаж болдог. Суурь загвар нь илүү их тооцоолол шаардаж, мөн хоцрогдол өндөр байдаг учраас үр ашигтай дасан зохицуулалт хийх шаардлагатай байдаг. Зурган дээрхээр хүсэлтийг ангилахад хямд агент болох RAG, чиглүүлэгч ашиглаж мөн эдгээр нь бусад үнэтэй агент руу шилжүүлж болно. Мэдээллийн аюулгүй байдлын үүднээс хиймэл оюуны оролт эсвэл гаралтыг хамгаалж болдог.

1.9 Ижил төстэй системүүдийн судалгаа

Хиймэл оюун агентийг нэвтрүүлсэн хэд хэдэн үйлдвэрийн түвшний шийдлүүд байдаг. Эдгээр системүүдтэй харьцуулалт хийснээр энэ судалгааны санал болгож буй зохиомжийн онцлог байр суурийг тодорхойлно [23] [22] [24].

Inngest нь serverless workflow платформ бөгөөд үйл явдлаар удирдагдах архитектур (Event-Driven Architecture) суурилсан, хэрэглэгчийн функцийн түвшинд ажилладаг. LLM интеграци байдаг ч RAG систем дутмаг байна. Голдуу startup, жижиг багуудад зориулагдсан.

Temporal нь даалгаврын төлөв байдлаа хадгалж, унасан ч яг өмнөх төлөв дээр буцан сэргэж чаддаг платформ бөгөөд олон хэлний дэмжлэгтэй. Гэвч тодорхой процессийг л гүйцэтгэх тул өөрийгөө удирдах агент болж чадахгүй. Uber, Netflix, Stripe зэрэг томоохон компаниуд ашигладаг.

Camunda нь BPMN 2.0 стандартад суурилсан процессын автоматжуулалтын платформ. Enterprise түвшний шийдэл боловч хиймэл оюуны дэмжлэг хязгаарлагдмал.

Дээрх системүүдээс ялгаатай нь энэ судалгааны санал болгож буй зохиомж нь хиймэл оюун агентуудыг анхнаасаа салангид микросервис болгон хөгжүүлж, төлөвлөгч нь микросервис бүр рүү чиглүүлж, Kafka-Flink ашиглан бодит цагийн урсгал боловсруулалт хийж, Temporal шиг дахин сэргэх чадвартайгаар нээлттэй эхийн технологи ашиглаж, хэвтээгээр өргөжих боломжтой байхаар зохиомжилсон. Энэ нь байгууллагын дотоод системд бие даасан байдлаар ажиллах боломжтой юм.

1.10 Бүлгийн дүгнэлт

Энэхүү бүлэгт хиймэл оюуны инженерчлэлийн үндсэн ойлголтуудыг авч үзлээ. Хиймэл оюуны инженерчлэл нь бэлтгэгдсэн суурь загвар дээр програм хангамж хөгжүүлэхэд чиглэсэн шинэ салбар юм [1]. Суурь загварын хөгжлийн замнал нь хэл загвараас том хэлний загвар, улмаар суурь загвар руу шилжих үйл явц байсан. Өөрийгөө удирдсан сургалт, Transformer архитектур, дараах сургалт аргууд нь өнөөгийн хүчирхэг хиймэл оюун системүүдийн суурь болсон.

Промпт инженерчлэл нь загварыг дасан зохицуулах хамгийн хялбар арга бөгөөд машин сургалт хийхгүйгээр программ хангамжид хиймэл оюун интеграц хийх боломжийг олгодог. RAG систем нь загварын дотоод мэдлэгийн хязгаарлалтыг даван, гадаад эх сурвалжаас мэдээлэл авч хариултын найдвартай байдлыг нэмэгдүүлдэг.

Хиймэл оюун агент нь өөрийн орчныг мэдрэх, түүн дээр үйлдэл хийх чадвартай систем юм. Агентын гурван гол бүрэлдэхүүн нь орчин, үйлдэл, даалгавар юм. Олон агентын зохиомж нь нарийн төвөгтэй бизнес процессуудыг шийдэхэд илүү тохиромжтой бөгөөд агент бүр мэргэшсэн даалгавартай байснаар системийг өргөжүүлэх боломжтой болгодог.

Inngest, Temporal, Camunda зэрэг одоо байгаа системүүдтэй харьцуулахад энэ судалгааны санал болгож буй зохиомж нь хиймэл оюун агентуудад зориулан тусгайлан бүтээгдсэн, Kafka-Flink суурилсан, нээлттэй эхийн технологи ашигласан онцлогтой.

2. МИКРОСЕРВИС БА АГЕНТУУД

Энэхүү бүлэгт микросервис архитектурын онолыг судлах ба үйл явдлаар удирдагдах архитектур (EDA), Apache Kafka, Apache Flink технологиудыг тайлбарлаж, агентуудыг микросервис болгон хөгжүүлэх үндэслэлийг тодорхойлно.

2.1 Монолитоос микросервис рүү

2.1.1 Монолитын эрин үе

Вэб аппликейшн хөгжүүлэлтийн эхэн үед гол архитектурын зохиомж нь монолит байв. Бүх бизнес логик, өгөгдлийн логикууд нэг том, нэгдсэн кодын санд амьдардаг байв. Программ хангамжийн хөгжүүлэгчид монолитуудыг хөгжүүлэхэд, нэвтрүүлэлт хийхэд энгийн, хялбар байв. [26]

2.1.2 Монолитыг өргөжүүлэх сорилт

Гэвч аппликейшнууд томрох тусам асуудлууд нэмэгдсээр ирсэн. Монолитыг өргөжүүлэх нь бүх зүйлийг нэгтгэсэн байдлаар өргөжүүлдэг. Нэг модульд хийсэн жижиг өөрчлөлт аливаа кодын санд орсноор энэ нь шинэчлэлтийг удаашруулж, унах эрсдлийг нэмэгдүүлдэг. Өөр өөр дэд процесс дээр ажиллаж байгаа багууд бие биенийхээ кодод хамаарал бүхий харилцаагаар холбогдож, хөгжлийг удаашруулж, алдаа гарах эрсдлийг нэмэгдүүлсээр ирсэн.

Монолит архитектураас эхэлсэн компаниуд эдгээр асуудлууд тулгарж хурдан үйл ажиллагаагаа явуулахын тулд багууд системийг шинэчлэх хүртэл хүлээх шаардлагатай байв. Энэ нь компанийн бизнесийн үйл ажиллагаанд маш том саад бэрхшээл болов.

2.1.3 Микросервис рүү шилжих

Эдгээр хязгаарлалтаас ангижрахын тулд компаниуд микросервис архитектур руу шилжиж эхлэв [26]. Энэ өөрчлөлт нь багуудад салангид байдлаар өөрчлөлтийг

хурдан байршуулалт хийх, аппликейшнийг дахин байршуулалт хийхгүйгээр шинэчлэл гаргах боломжийг олгосон. Микросервис рүү шилжих нь зөвхөн өргөжүүлэх чадварыг сайжруулаад зогсохгүй, багуудад бие даасан байдал өгч, хамтын ажиллагааны ачааллыг бууруулж, инновацийг хурдасгасан.

2.1.4 Микросервисийн тодорхойлолт

Микросервис архитектур нь програм хангамжийн гол зохиомжийн аргуудын нэг бөгөөд аппликейшныг жижиг, бие даасан сервисүүдэд хувааж, өргөтгөхөд хялбар байдлаар хөгжүүлдэг. Энэхүү архитектурын гол онцлог нь дөрвөн чухал шинж чанарт илэрхийлэгддэг. [26].

Эхний шинж чанар нь бие даасан байдал юм. Микросервис бүр өөрийн тодорхой үүрэгтэй бөгөөд шаардлагатай бол өөрийн өгөгдлийн сантай байдаг. Энэ нь сервис бүр бусад сервисээс хараат бус ажиллах боломжийг олгодог. Хоёр дахь шинж чанар нь уян хатан хөгжүүлэлт юм. Өөр өөр багууд өөр өөр технологи, програмчлалын хэл ашиглан өөрсдийн сервисийг хөгжүүлж болдог. Энэ нь багууд өөрсдийн мэргэжлийн чиглэлд тохирсон технологи сонгох чөлөөг өгдөг. Гурав дахь шинж чанар нь өргөжих чадвар юм. Бүх системийг өргөжүүлэх шаардлагагүй бөгөөд зөвхөн ачаалал их эсвэл илүү их нөөц шаарддаг сервисийг л өргөжүүлэх боломжтой. Дөрөв дэхь шинж чанар нь найдвартай байдал юм. Хэрэв нэг сервис алдаа гарч унавал бусад сервисүүд хэвийн ажиллаж үргэлжлэх бөгөөд энэ нь системийн ерөнхий тогтвортой байдлыг хангадаг.

2.2 Микросервисийн давуу тал

Микросервис архитектур нь монолит системээс олон талаараа давуу талтай байдаг. Технологийн олон янз байдлын талаас авч үзвэл, сервис бүр өөрийн хэрэгцээнд хамгийн тохирсон технологи сонгох боломжтой. Жишээлбэл, нэг сервис Python програмчлалын хэл ашиглаж өгөгдөл боловсруулалт хийж болох бол нөгөө сервис Go ашиглан өндөр гүйцэтгэлтэй серверийн хэсэг хариуцаж, өөр нэг сервис Node.js ашиглан бодит цагийн холболт зэргийг удирдаж болно. [27].

Багуудын бие даасан ажиллагааны хувьд баг бүр өөрийн сервисийг хараат бусаар хөгжүүлж, шууд хэрэглээнд нэвтрүүлэх чадвартай. Энэ нь багуудын хурд, уян хатан байдлыг нэмэгдүүлдэг. Хурдан нэвтрүүлэлтийн талаас авч үзвэл том системийг бүхэлд нь дахин нэвтрүүлэх шаардлагагүй бөгөөд зөвхөн өөрчлөлт орсон сервисийг л нэвтрүүлэхэд цаг хугацааг ихээхэн хэмнэдэг. Илүү сайн өргөжих чадварт ачаалал их байгаа тодорхой сервисийг л өргөжүүлэх нь бүх системийг бүхэлд нь өргөжүүлэхээс илүү үр ашигтай бөгөөд зардал хэмнэлттэй. Эцэст нь алдааны тусгаарлалтын талаас авч үзвэл нэг сервисийн алдаа нь бусад сервист шууд дамжихгүй тул системийн бусад хэсэг хэвийн үргэлжлэн ажилладаг.

2.3 Микросервисийн сорилтууд

Микросервис архитектур олон давуу талтай боловч практикт тулгарах сорилтууд ч багагүй байдаг. Нарийн төвөгтэй байдлын хувьд олон сервисүүдийг зэрэг удирдах, тэдгээрийн харилцааг хянах, байршуулалтыг хийх нь монолит системээс илүү төвөгтэй бөгөөд тусгай хяналтын хэрэгслүүд шаарддаг. [27].

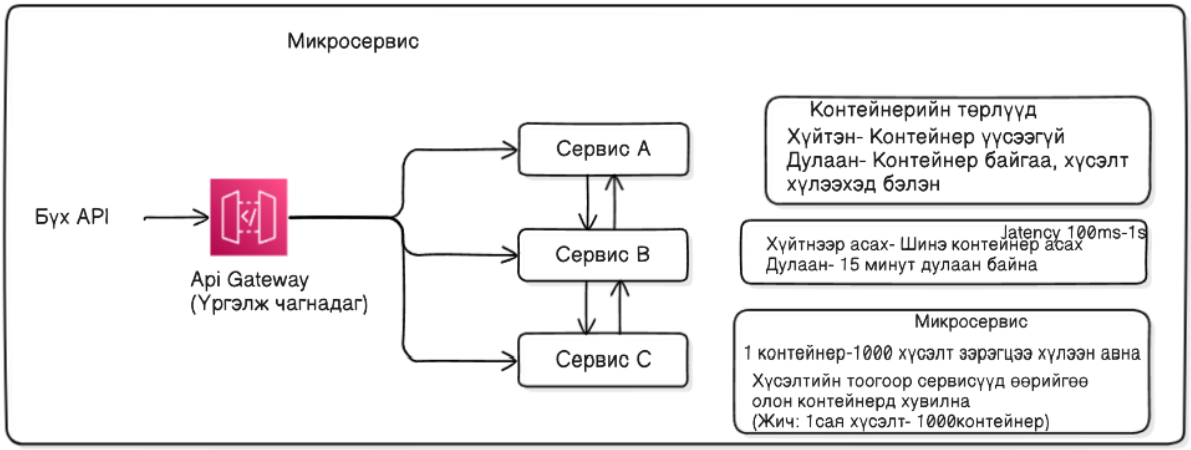
Өгөгдлийн тогтмол байдлын асуудал нь сервис бүр өөрийн өгөгдлийн сантай байдгаас үүсдэг. Олон сервисийн өгөгдлийн нийцтэй, уялдаатай байдлыг хангах нь уламжлалт транзакцийн удирдлагаар шийдэгдэхгүй асуудал болдог. Сүлжээний хоцрогдлын талаас авч үзвэл сервисүүд хоорондоо сүлжээгээр харилцдаг учир нэмэлт хоцрогдол гардаг бөгөөд энэ нь системийн ерөнхий гүйцэтгэлд нөлөөлдөг. Алдаа илрүүлэх хэцүү байдал нь олон сервисүүдээр дамжин явах хүсэлтийн алдааг олж тодорхойлох, засах ажил үйл ажиллагаа төвөгтэй болдог. Сервис хоорондын харилцааны асуудал нь сервисүүд хэрхэн үр дүнтэй харилцах, ямар протокол ашиглах, өгөгдлийн формат хэрхэн нийцүүлэх зэрэг олон нарийн асуудлыг шийдэхийг шаарддаг. Эцэст нь транзакцийн удирдлагын асуудал нь олон сервисүүдээр транзакци явуулах нь өгөгдлийн сангууд түгжигдэх, тогтворгүй байдал үүсэх эрсдлийг нэмэгдүүлдэг.

2.4 Микросервис хоорондын харилцаа

Микросервисүүд хоорондоо хоёр гол аргаар харилцдаг: [27].

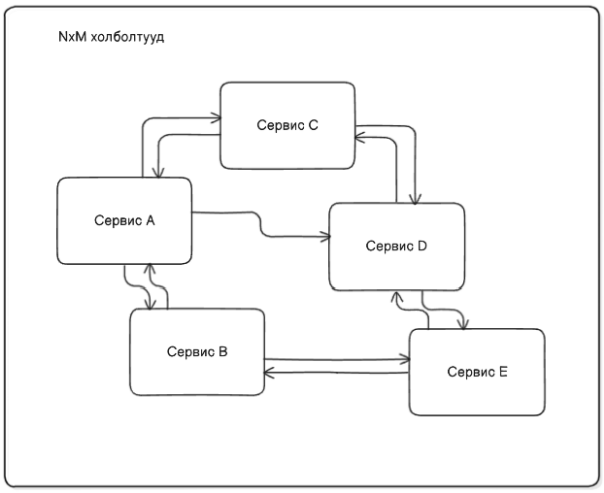
2.4.1 Синхрон харилцаа

Синхрон харилцаа нь HTTP REST програмчлалын интерфэйс эсвэл gRPC ашиглан шууд хүсэлт илгээж хариу хүлээдэг арга юм. Энэ арга нь хэрэгжүүлэхэд харьцангуй энгийн боловч сул талтай. Эхний сул тал нь нягт хамаарал буюу tight coupling үүсгэдэг. Сервисүүд бие биенээсээ шууд хамаарч байдаг учир нэг сервис өөрчлөгдөх үед бусад сервист нөлөөлдөг. Хоёр дахь сул тал нь нэг сервис унавал түүнээс хамааралтай бусад сервисүүд ч гэсэн алдаа гаргаж зогсдог. Энэ нь системийн найдвартай байдлыг бууруулдаг. Гурав дахь сул тал нь хоцрогдол нэмэгддэг. Сервисүүд бие биенээсээ хариу хүлээж байдаг учир хариултын цаг удаан байх тусам ерөнхий системийн хоцрогдол нэмэгддэг.



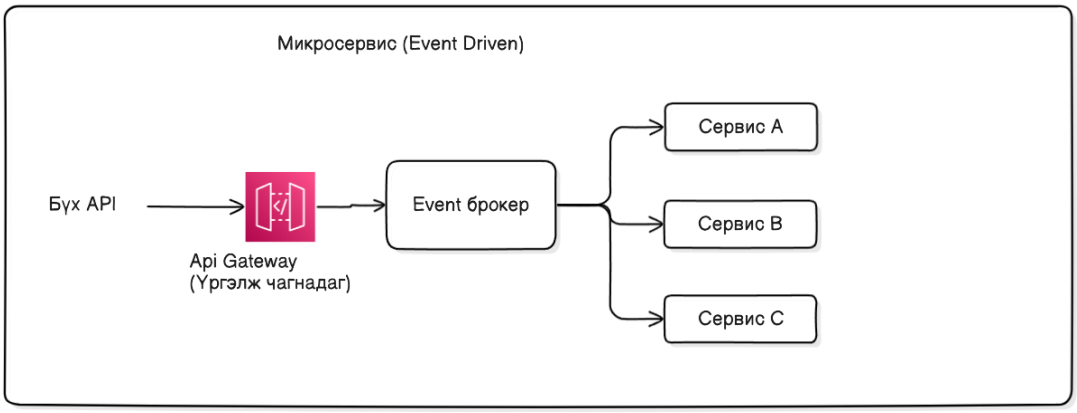
Зураг 2.1: Синхрон микросервис

Сервисийн тоогоор NxM холболтын харьцаагаар холболт нэмэгдэх учир үүнийг найдвартай удирдахад маш хүндрэлтэй болно.



Зураг 2.2: Синхрон микросервисийн NxM холбоо

2.4.2 Асинхрон харилцаа

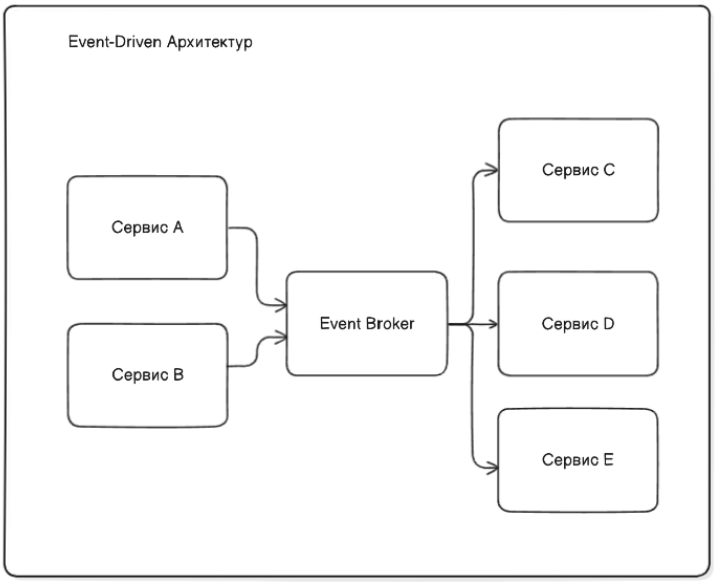


Зураг 2.3: Асинхрон микросервис

Асинхрон харилцаа нь мессежийн дараалал ашигладаг арга бөгөөд RabbitMQ, Apache Kafka зэрэг технологиудыг ашиглан мессеж солилцдог. Энэхүү аргууд дараах онцлогтой. Эхний онцлог нь салангид байдлыг үүсгэдэг. Сервисүүд шууд бус мессежийн брокероор дамжин харилцдаг учир бие биенээсээ хараат бус байдаг. Хоёр дахь онцлог нь илүү найдвартай байдаг. Нэг сервис түр зуур унасан ч мессеж хадгалагдсан байх тул дараа нь боловсруулагдах боломжтой. Гурав дахь онцлог нь синхрон харилцаанаас илүү төвөгтэй хэрэгжилттэй байдаг. Мессежийн формат тодорхойлох, алдааны менежмент хийх, мессежийн дараалал хангах зэрэг нэмэлт асуудлыг шийдэх шаардлагатай.

2.5 Үйл явдлаар удирдагдах архитектур буюу EDA

2.5.1 EDA гэж юу вэ



Зураг 2.4: Үйл явдлаар удирдагдах архитекурт микросервис

Микросервисүүд гарч ирснээр шинэ сорилт бий болсон [27].: эдгээр сервисүүд хэрхэн үр дүнтэй харилцах вэ? Хэрэв бид сервисүүдийг шууд gRPC эсвэл API-ээр холбовол бид асар том хамаарлын сүлжээ үүсгэнэ. Хэрэв нэг сервис унавал энэ нь холбогдсон замын дагуух бүх node-д нөлөөлнө. Мөн гацалт маш ихээр үүснэ.

EDA нь энэхүү асуудлыг шийдэж болдог. Нягт холбогдсон, синхрон харилцааны оронд EDA нь бүрэлдэхүүн хэсгүүдэд үйл явдлаар асинхрон харилцах боломжийг олгодог. Сервисүүд бие биенийгээ хүлээхгүй. Бодит цагт юу болж байгааг мэдээд хариу үйлдэл үзүүлдэг.

2.5.2 EDA-ын давуу тал

EDA нь дөрвөн гол давуу талтай. Салангид байдлын хувьд сервисүүд үйл явдлаар харилцдаг учир тэд бие биенээсээ бүрэн хараат бус байдаг. Нэг сервис өөрчлөгдөх эсвэл түр зуур унах нь бусад сервист шууд нөлөөлөхгүй бөгөөд энэ нь системийн уян хатан чанар байдлыг дээшлүүлдэг.

Өргөжүүлэх чадварын талаас авч үзвэл сервис бүр үйл явдлыг бие даан боловсруулдаг учир шаардлагатай сервисийг л өргөжүүлэх боломжтой. Энэ нь бүх системийг өргөжүүлэхээс хамаагүй хялбар бөгөөд зардал хэмнэлттэй. Уян хатан байдлын хувьд шинэ сервис нэмэх эсвэл одоо байгаа сервисийг өөрчлөх нь бусад сервист өөрчлөлт шаардахгүй. Үйл явдлын формат өөрчлөгдөөгүй бол өөр бусад сервисүүд хэвээр ажилладаг. Эцэст нь бодит цагийн боловсруулалтын талаас авч үзвэл үйл явдлууд тэр даруйдаа боловсруулагдах тул систем нь өөрчлөлт, шинэ мэдээлэлд маш хурдан хариу үйлдэл үзүүлэх чадвартай.

2.5.3 Apache Kafka

Apache Kafka нь салангид, өндөр дамжуулалттай, бага хоцрогдолтой EDA-ийн төв мэдээллийн систем болж чаддаг. [20].

Kafka-ийн үндсэн ойлголтууд

Apache Kafka-ийн архитектур нь таван үндсэн ойлголт дээр суурилдаг. [20]. Сэдэв буюу topic нь үйл явдлуудыг ангилах логик сувагийн үүрэг гүйцэтгэдэг. Жишээлбэл ”user-events”, ”order-events” гэх мэт нэршлээр үйл явдлуудыг ангилж хадгалдаг. Үйлдвэрлэгч буюу producer нь үйл явдлыг topic руу бичдэг аппликейшн юм. Хэрэглэгч буюу consumer нь эсрэгээр сэдвээс үйл явдлыг уншиж боловсруулдаг аппликейшн болно. Хэсэглэл буюу partition нь topic-ийг өргөжүүлэх, параллель боловсруулалт хийх боломжийг олгодог механизм юм. Нэг topic олон partition-д хуваагдаж, параллель уншигдаж боловсруулагдах боломжтой. Эцэст нь хэрэглэгчийн груп буюу consumer group нь олон consumer нэг баг болон зохион байгуулагдаж ачааллыг хуваарилан боловсруулах боломжийг олгодог.

Kafka-ийн давуу тал

Apache Kafka дараах давуу талуудтай. [20] Хэвтээ өргөжих чадварын хувьд Kafka-ийн салангид зохиомж нь саадгүйгээр шинэ агент эсвэл хэрэглэгч нэмэх боломжийг олгодог. Системийн ачаалал нэмэгдэх тусам partition нэмж, хэрэглэгч нэмж өргөжүүлэх нь маш энгийн.

Бодит цагийн үйл явдал боловсруулалт нь агентуудад өөрчлөлтөд шууд хариу үйлдэл үзүүлэх боломжийг олгодог. Мессеж миллисекундын дотор дамжих учир бодит цагийн систем бүтээхэд тохиромжтой. Салангид хамааралын талаас авч үзвэл сэдвүүдээр харилцах нь агентууд хараат бус, өргөжүүлэх боломжтой байхыг баталгаажуулдаг. Агент нэмэх, хасах, өөрчлөх нь бусад агентад нөлөөлөхгүй. Үйл явдлын хадгалалтын талаас авч үзвэл тогтвортой мессежийн хадгалалт нь өгөгдөл дамжилтын явцад алга болохгүй гэдгийг баталгаажуулдаг. Мессеж диск дээр хадгалагддаг учир хэрэглэгч алдаатай байсан ч мессеж хадгалагдсан байна. Эцэст нь дахин ачааллуулж гүйцэтгэх боломжийн хувьд Kafka нь салангид лог ашигладаг учраас үйл явдал бүр хадгалагдаж, алдаа засах, үнэлгээ хийх, загвар дахин сургахад дахин тоглуулж ашиглах боломжтой.

2.5.4 Apache Flink

Apache Flink нь сүүлийн төлвөөр тооцоолол, үйл явдал цагийн боловсруулалт хийх чадвартай салангид урсгал тооцооллын фреймворк юм. [21] Flink нь Kafka-тай хамт ашиглах замаар хүчирхэг бодит цагийн өгөгдөл боловсруулалтын систем бүтээх боломжийг олгоно.

Flink-ийн давуу тал

Apache Flink нь дараах давуу талуудтай. Flink нь төлөв байдлыг найдвартай удирдаж, нарийн төвөгтэй тооцоолол хийх боломжийг олгодог. Үйл явдлуудын хоорондох хамаарлыг хадгалж, өөр бусад цар хүрээтэй өгөгдлийг ашиглан нарийн шинжилгээ хийж чаддаг. Үйл явдлын цаг дээр үндэслэн бодит цагийн шинжилгээ хийх чадвартай. Энэ нь хоцрогдсон мессежүүдийг зөв цагийн дагуу боловсруулах боломжийг олгодог. Flink нь секундэд сая сая үйл явдлыг боловсруулах чадвартай бөгөөд параллель боловсруулалт ашиглан асар их хэмжээний өгөгдлийг боловсруулж чаддаг. Эцэст нь ”яг ганц” семантикийн талаас авч үзвэл мэдээлэл яг нэг удаа л боловсруулагдахаар баталгааждаг. Энэ нь өгөгдөл алдагдах эсвэл давхардах асуудлыг шийддэг.

2.6 Flink ба хиймэл оюун

Flink нь том хэлний загвартай ажиллах чадвартай. [17] Flink нь өгөгдлийг авч, том хэлний загвар руу илгээж, хариу авах боломжийг олгодог. Энэ нь төлөвлөгч агентыг Flink аппликейшн болгон хөгжүүлэх боломжийг олгодог.

Ажиллах зарчмыг жишээгээр авч үзье. Эхлээд Kafka сэдвээс үйл явдлыг уншина. Дараа нь том хэлний загвар ашиглан цар хүрээг ойлгох, даалгаврыг задлах, төлөвлөгөө гаргах үйл явцыг гүйцэтгэнэ. Үр дүнг өөр Kafka topic руу бичинэ. Эцэст нь бусад агентууд энэ үр дүнг уншиж тодорхой үүргүүдээ гүйцэтгэнэ. Иймээс рекурисв байдлаар ажиллаж чаддаг. Энэхүү зохион байгуулалт нь урсгал боловсруулалтын давуу талыг хиймэл оюун агенттай нэгтгэж чаддаг.

Flink болон RAG

Төөрөгдлийг багасгахын тулд том хэлний загварыг бодит өгөгдөлд суурилуулах хэрэгтэй. Flink нь хайлтаар нэмэгдүүлсэн үүсгэлтийн зохион байгуулалтыг бүтээхэд чухал үүрэг гүйцэтгэдэг. [18]

Энэ үйл явц нь таван үндсэн алхамтай. Эхлээд Flink нь өгөгдлийг боловсруулж, цэвэрлэж, хувиргана. Дараа нь загварын гаралтыг ашиглан өгөгдлийг embedding болгон хөрвүүлнэ. Гурав дахь алхамд векторчилсан хэсгийн төлөөлөл буюу embedding-г Kafka topic руу бичиж хадгална. Дөрөв дэхь алхамд Kafka Connect ашиглан векторчилсан хэсгийн төлөөллийг векторын өгөгдлийн сан руу синхрончилдог. Эцсийн алхамд агентууд хайлтаар нэмэгдүүлсэн үүсгэлт ашиглан бодит өгөгдөл дээр суурилсан найдвартай хариулт өгөх боломжтой болно. Энэ урсгал нь бодит цагт өгөгдлийг боловсруулж, том хэлний загварын мэдлэгийг тасралтгүй шинэчилж байх боломжийг олгодог.

2.6.1 Агентууд ба EDA

Хиймэл оюун агентуудыг өргөжүүлэх нь үндсэндээ салангид систем байж болох талаарх асуудал юм. Агентууд нь шийдвэр гаргаж, үйлдэл хийхийн тулд олон эх сурвалж, бусад агентууд, хэрэглүүрүүд, гадаад системүүдээс мэдээлэл цуглуулах шаардлагатай. [1]

Агентууд яагаад үйл явдлаар удирдагдах архитектур шаарддаг вэ гэдэг нь гурван гол шалтгаанаас үүдэлтэй. Асинхрон шинж чанарын хувьд агентууд хүн шиг ажилладаг. Агент нь олон эх сурвалжаас мэдээлэл цуглуулж, өгөгдлийг шинжилж, бүх талын мэдээлэлд үндэслэн шийдвэр гаргах хэрэгтэй. Эдгээр үйл явцууд нь асинхрон шинжтэй бөгөөд тодорхой дарааллаар биш параллель явагддаг. [17]

Агентууд нь өмнөх үйлдлүүдийн үр дүн, бусад агентуудын гаралт, хэрэглэгчийн түүх зэрэг олон мэдээллийг нэгтгэж ашигладаг. Энэ нь хамаарлыг оновчтой удирдах, бодит цагийн өгөгдлийн урсгалыг гаргах онцгой шаардлагыг бий болгоно. Агентын гаралт нь зөвхөн хиймэл оюунт аппликейшн рүү биш, харин өгөгдлийн сан, CRM, харилцагчийн төлбөр тооцоо зэрэг бусад чухал системүүд рүү урсаж болно. Мөн алдаа засах, шинжилгээ хийх зорилгоор лог хадгалах шаардлагатай.

Агентуудыг синхрон холболттой gRPC болон API-аар холбож болно, гэвч энэ нь нягт холбогдсон системүүдийг бий болгодог. Энэхүү нягт холбоос нь өргөжүүлэх, дасан зохицох, эсвэл ижил өгөгдлийн олон хэрэглэгчдийг дэмжихэд хэцүү болгодог. Агентууд нь уян хатан байдлыг шаарддаг. Тэдний гаралт нь бусад агентууд, сервисүүд, платформуудад тодорхой дүрмийн дагуу үр дүн дамжих ёстой.

2.7 Бүлгийн дүгнэлт

Энэхүү бүлэгт микросервис архитектурын онол, практик, түүнчлэн үйл явдлаар удирдагдах архитектурын давуу талыг дэлгэрүүлэн судалсан. Монолит системээс микросервис рүү шилжих нь программ хангамжийн хөгжлийн чухал дэвшил болов. [17]

Микросервис хоорондын харилцаа нь хоёр гол аргаар хэрэгждэг. Синхрон харилцаа нь HTTP REST эсвэл gRPC ашигладаг боловч нягт хамаарал үүсгэж, нэг сервис унавал бусад сервисүүдэд алдаа гаргадаг. Сервисийн тоогоор NxM холболтын нарийн төвөгтэй

байдал нь системийг удирдахад хүндрэлтэй болгодог. Асинхрон харилцаа нь мессежийн брокер ашигладаг бөгөөд салангид байдлыг бий болгож, найдвартай боловч илүү төвөгтэй хэрэгжилттэй.

EDA нь микросервисийн энэхүү харилцааны хамгийн үр дүнтэй шийдэл болдог. Нягт холбогдсон, синхрон харилцааны оронд EDA нь бүрэлдэхүүн хэсгүүдэд үйл явдлаар асинхрон харилцах боломжийг олгодог. Салангид байдал, өргөжүүлэх чадвар, уян хатан байдал, бодит цагийн боловсруулалт зэрэг давуу талууд нь EDA-г орчин үеийн микросервис архитектурын суурь болгосон.

Apache Kafka нь EDA-ын хүчирхэг хэрэглүүр ба ”Topic, producer, consumer, partition, consumer group” зэрэг үндсэн ойлголтууд нь системийг өргөжүүлэх, параллель боловсруулалт хийх боломжийг олгодог. [18] Kafka-ийн хэвтээ өргөжих чадвар, бага хоцрогдол, салангид байдал, үйл явдлын хадгалалт, дахин тоглуулах боломж зэрэг онцлогууд нь өргөн ашиглагддаг шалтгаан болов.

Apache Flink нь Kafka-тай хамт ашиглахад илүү хүчирхэг болдог. Өгөгдөл холбож дамжуулах боловсруулалт, өндөр дамжуулалт, ”яг л нэг” зарчим зэрэг давуу талууд нь нарийн төвөгтэй урсгал боловсруулалтад тохиромжтой. Flink нь хиймэл оюуны загвартай холбогдож, төлөвлөгч агентыг Flink app болгон хөгжүүлэх боломжийг олгодог. Flink болон RAG-ийг хослуулах нь төөрөгдлийг багасгаж, бодит өгөгдөлд суурилсан хиймэл оюун систем бүтээхэд тусалдаг.

Хиймэл оюун агентууд бүр өөр өөрийн гэсэн даалгаврын цар хүрээ хариуцах учраас микросервис шиг салангид байдлаар ашиглагдвал цаашдын хиймэл оюунт программ хөгжүүлэхэд үр дүнтэй. Агентуудын олон цар хүрээт мэдээллийн хамаарлаар, олон хэрэглэгчдэд үйлчлэхэд EDA зайлшгүй шаардлагатай болгодог. Агентуудыг gRPC эсвэл API-аар холбох нь боломжтой боловч нягт холбоос үүсгэж, өргөжүүлэх, дасан зохицоход хүндрэлтэй болгодог. EDA нь агентуудыг салангид микросервис болгон хөгжүүлэх, өргөжүүлэх, найдвартай байлгах хамгийн тохиромжтой арга замуудын нэг болох юм.

Энэхүү бүлгээс харахад микросервис архитектур нь монолитын асуудлуудыг шийдэж чадсан боловч шинэ сорилтууд авчирсан. Үйл явдлаар удирдагдах архитектур, Kafka, Flink зэрэг технологиуд нь эдгээр сорилтуудыг шийдэж, илүү уян хатан, өргөжих

боломжтой, найдвартай систем бүтээх суурийг бүрдүүлсэн. Дараагийн бүлэгт эдгээр онолуудыг ашиглан тодорхой асуудлуудыг хэрхэн шийдэх талаар авч үзнэ.

3. САНАЛ БОЛГОЖ БУЙ ЗОХИОМЖ

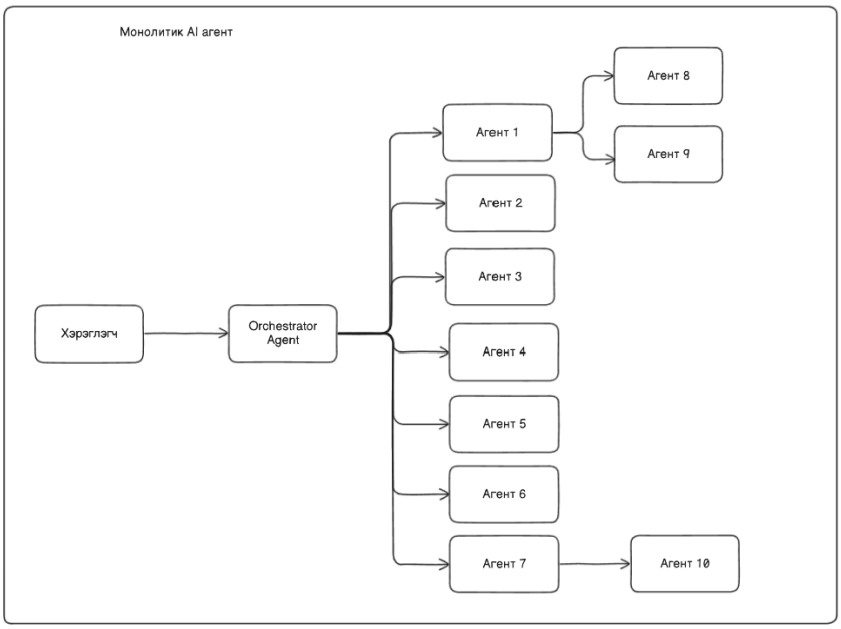
Энэхүү бүлэгт олон агентын системийг монолит болон микросервис архитектураар нэвтрүүлэхэд тулгарах асуудлуудыг тодорхойлно. Дараа нь үйл явдлаар удирдагдах архитектур (EDA) ашиглан агентуудыг салангид микросервис болгон хөгжүүлэх зохиомжийг санал болгоно.

3.1 Монолит агентуудын эрсдэл

Хиймэл оюун хөгжүүлэхэд тогтсон стандарттай фреймворк байдаггүй. [16]. Хиймэл оюуны анхны фреймворкууд нь туршилт хийхэд зориулагдсан бөгөөд бизнесийн бодит орчинд зориулагдаагүй байдаг. Хөгжүүлэгчид notebook дээр загвар туршиж, тодорхой хязгаарлагдмал ажлын урсгалыг ажиллуулж байсан. Гэвч бодит орчинд агентууд нь notebook-д амьдрах нь зохимжгүй. Жишээлбэл өгөгдөл нь хуучрах, өөр дэд процессийн үйл ажиллагаа өөрчлөгдвөл алдаа гарах, нэг газар унавал алдаа барихад хэцүү зэрэг асуудал тулгарсан. Тиймээс агентууд бодит орчинд ажиллаж, бизнесийн үйл ажиллагаатай интеграц хийж, найдвартайгаар өргөжих шаардлагатай байдаг.

3.1.1 Нягт холбогдсон монолит агентуудын архитектур

Монолит агентын систем нь бүх агентууд нэг аппликейшн доторх модуль эсвэл класс болж ажилладаг. [16]. Хэрэглэгч Orchestrator агент руу хүсэлт илгээхэд Orchestrator нь бусад агентуудыг API-аар дуудах ба агентууд хоорондоо NxM нягт холбоостой байдаг. Энэ нь хялбар харагдах ч хэд хэдэн ноцтой асуудал үүсгэдэг.



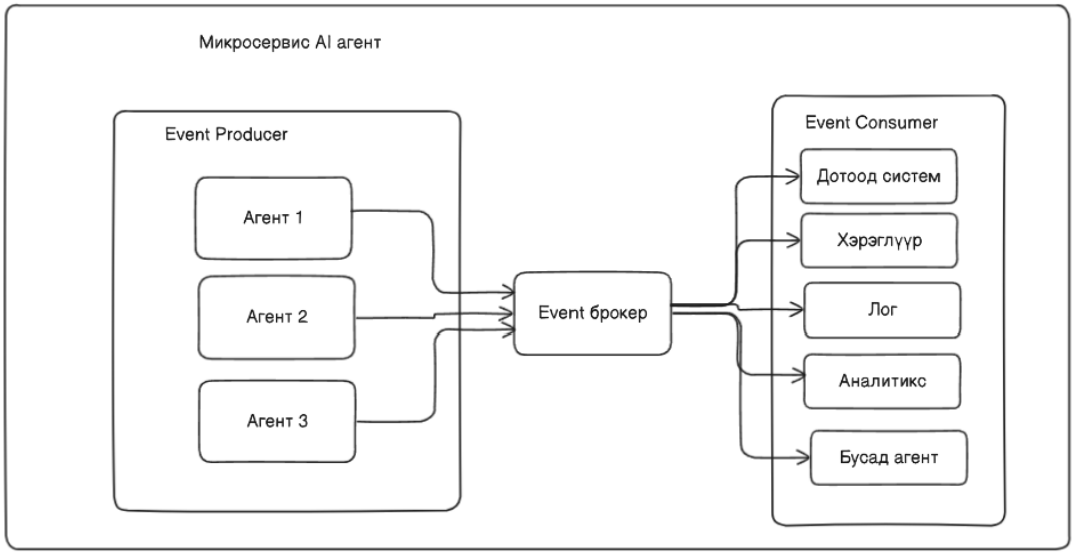
Зураг 3.1: Монолит агентын архитектур: NxM нягт холбоос

Зураг 3.1-т монолит агентын архитектурыг дүрсэлсэн. Зураг 3.1-аас харахад агентууд хоорондоо NxM нягт холбоостой байна. Энэ архитектурын дараах сул талууд байна:

- Агентууд бие биенээсээ шууд хамааралтай байдаг учир нэг агентын өөрчлөлт нь бусад агентуудад нөлөөлдөг
- Бүх агентууд нэгэн зэрэг өргөжих ёстой бөгөөд тусдаа өргөжүүлэх боломжгүй
- Нэг агентыг шинэчлэхэд бүх системийг дахин суурилуулах шаардлагатай
- Нэг агент унах нь бүх системийг доголдуулах магадлалтай (дамжин унах алдаа)

3.2 Микросервис агентуудын шийдэл

Эдгээр асуудлыг шийдэхийн тулд микросервис архитектурт ашигладаг EDA нь хиймэл оюуны найдвартай нэвтрүүлэлтийн аргачлал болдог. NxM нягт холбоосын оронд агентууд эвент брокероор дамжуулан бие биетэйгээ мессежээр харилцдаг. Энэ нь хамааралыг N+M болгон бууруулж, агентуудыг бүрэн салангид болгодог.



Зураг 3.2: Үйл явдлаар удирдагдах олон агентын систем

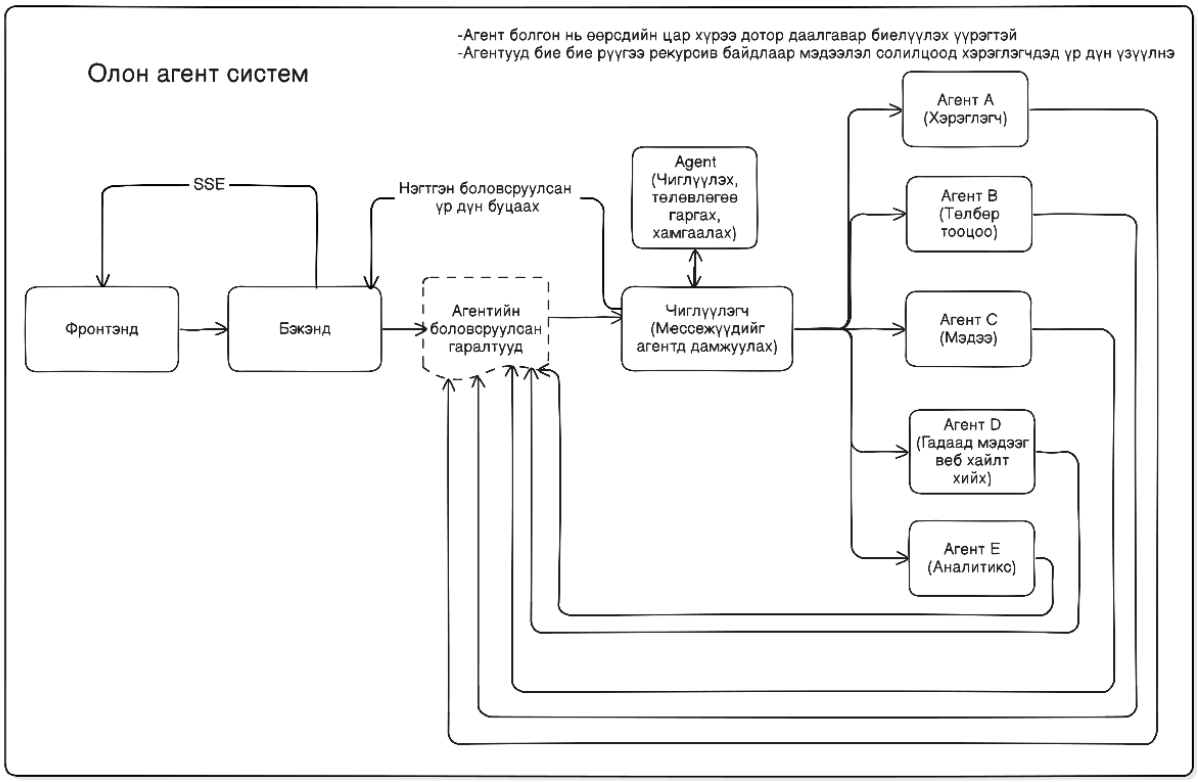
Зураг 3.2-т үйл явдлаар удирдагдах олон агентын системийг дүрсэлсэн. Зураг 3.2-аас харахад систем нь гурван гол бүрэлдэхүүнтэй:

- **Эвент үүсгэгчүүд:** Өөр өөр зориулалттай агентууд (төлөвлөгч, чиглүүлэгч, мэдээ, төлбөр тооцооны гэх мэт) үйл явдал үүсгэж эвент брокер руу илгээдэг
- **Эвент брокер:** Apache Kafka, RabbitMQ зэрэг брокер нь агентуудын хоорондын төв мэдээллийн систем болж үйл явдлыг хадгалдаг
- **Эвент хүлээн авагч:** Агентууд болон бусад системүүд брокерээс үйл явдлыг хүлээн авч боловсруулна

Энэхүү архитектур нь монолит системийн бүх сул талыг шийддэг. Салангид байдлын хувьд агентууд эвент брокероор харилцах тул бие биенээсээ хараат бус байдаг. Нэг агент унах нь бусдад шууд нөлөөлөхгүй. Хэрэглүүр, системтэй холбогдсон байдал нь хуучраагүй бодит цагийн өгөгдөл авах зэрэг боломж нээдэг. Асинхрон харилцааны хувьд агентууд бие биенийхээ хариуг хүлээхгүй, параллель боловсруулалт хийх боломжтой. Бие даан өргөжүүлэх чадварын хувьд агент бүр өөрийн цар хүрээний хэрэгцээний тохирсон хэмжээгээр өргөжиж болно. Байршуулалтын салангид байдлын хувьд агент бүрийг бусдад нөлөөлөхгүйгээр тусдаа суурилуулж болно. Найдвартай байдлын хувьд үйл

явдлууд хадгалдаг учир агент түр зуур унасан ч мэдээлэл алдагдахгүй, дахин ачааллуулж болно. Иймээс ч бүх үйл явдал лог байдлаар хадгалагдах учир алдаа олж засах, тест хийх, аудит хийж чадна.

3.2.1 Микросервист суурилсан хиймэл оюун агентуудын зохиомжийн жишээ



Зураг 3.3: Санал болгож буй архитектур: Kafka-Flink суурилсан олон агентын систем

Зураг 3.3-т санал болгож буй архитектурыг дүрсэлсэн. Зураг 3.3-аас харахад систем нь дараах байдлаар ажиллана:

1. Хэрэглэгч фронтенд дээр хүсэлт илгээнэ
2. API Gateway хүсэлтийг хүлээн авч Kafka сэдэв рүү үйл явдал болгон илгээнэ
3. Зохион байгуулагч агент суурь загвараар хүсэлтийг ангилж, RAG-аас аль агент руу чиглүүлэхийг тодорхойлно
4. Өөр өөр зориулалттай агентууд (мэдлэгийн, хөрөнгө оруулалтын, мэдээний гэх мэт) өөрсдийн үүргийг гүйцэтгэнэ

5. Агентууд үр дүнгээ Kafka сэдэв рүү буцааж илгээнэ
6. API Gateway хариултыг SSE ашиглан бодит цагийн stream хэлбэрээр хэрэглэгч рүү хүргэнэ

Энэхүү архитектур нь дараах давуу талуудтай:

- **Параллель боловсруулалт:** Агент бүр бие даан ажиллаж, мянга мянган хүсэлтийг боловсруулна
- **Уян хатан өргөтгөл:** Шинэ агент нэмэх нь бусад агентын кодыг өөрчлөхгүй
- **Динамик шийдвэр гаргалт:** Хиймэл оюун дараагийн алхмыг динамикаар тодорхойлно
- **Лог хадгалалт:** Бүх үйл явдал хадгалагдаж, үнэлгээ, дахин сургалтад ашиглаж болно

3.3 Бүлгийн дүгнэлт

Энэхүү бүлэгт олон агентын системийг монолит болон микросервис архитектураар нэвтрүүлэхэд тулгарах асуудлуудыг тодорхойлж, үйл явдлаар удирдагдах архитектур ашиглан салангид, өргөжих боломжтой систем бүтээх шийдлийг санал болголоо.

Монолит агентын гол асуудлууд нь агентийн тоогоор NxM нягт холбоос үүссэнээр өргөжүүлэх хүндэрч, цаашлаад дамжин унах алдаа, хөгжүүлэлтийн удаашрал зэргийг үүсгэдэг. Эдгээр асуудлыг шийдэхийн тулд микросервис архитектурт EDA ашигласнаар агентууд бие биенээсээ салангид, асинхрон байдлаар найдвартай байдлаар ажиллах боломжтой болно. Inngest, Temporal, Camunda зэрэг одоо байгаа системүүдтэй харьцуулахад энэхүү судалгааны санал болгож буй зохиомж нь анхнаасаа хиймэл оюун агентуудыг салангид микросервис болгон хөгжүүлж, Kafka-Flink ашиглан бодит цагийн урсгал боловсруулалт хийж, нээлттэй эх суурилсан технологи ашиглаж, хэвтээгээр өргөжих зэрэг онцлогтой.

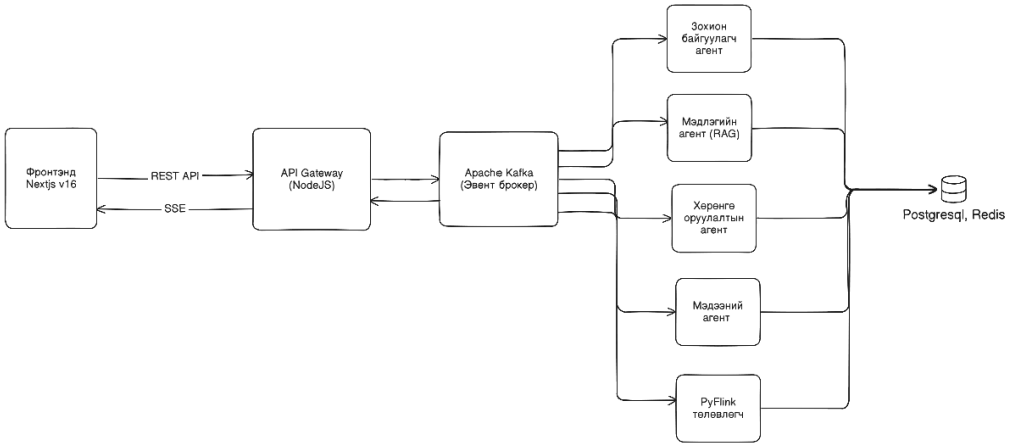
Уг судалгааны ажлын технологийн архитектур нь Apache Kafka-г агентуудын мэдээлэл солилцох суваг болгон ашиглаж салангид байдлыг хангаж, асинхрон харилцаагаар хүлээлт үүсгэхгүй, олон хэрэглэгч дэмжих, найдвартай байдлыг хангах, алдаатай хүсэлтийг дахин ачааллуулах зэрэг үйлдлүүдийг EDA-ын тусламжтайгаар хэрэгжүүлнэ. Энэ архитектурын зохиомж нь бие даасан ухаалаг микросервис болгон хөгжүүлэх боломжийг олгоно.

4. ХЭРЭГЖҮҮЛЭЛТ

Энэхүү бүлэгт өмнөх бүлгүүдэд судалсан онол, санал болгосон зохиомжийн дагуу демо системийг хөгжүүлж туршина. Эхлээд системийн архитектур, агентуудын үүргийг тайлбарлаад, дараа нь технологийн стек, хэрэгжүүлэлтийн явцыг дэлгэрэнгүй авч үзнэ.

Демо систем нь Монголын Хөрөнгийн Биржийн бодит өгөгдөлд суурилсан хувьцааны шинжилгээний веб платформ юм. Систем нь хиймэл оюун агентуудыг микросервис архитектурт EDA байдлаар нэвтрүүлж, хэрэглэгчдэд хувь хүнд тохирсон дүн шинжилгээ, зөвлөмж өгөх боломжийг олгоно.

4.1 Системийн архитектур



Зураг 4.1: Демо системийн архитектур

Зураг 4.1-т демо системийн архитектурыг дүрсэлсэн. Зураг 4.1-аас харахад систем нь дараах бүрэлдэхүүнүүдтэй:

- **Frontend:** Next.js 16 ашиглан хэрэглэгчийн интерфэйс хөгжүүлсэн
- **API Gateway:** HTTP/REST хүсэлтийг хүлээн авч, Kafka руу эвент хэлбэрээр дамжуулна

- **Apache Kafka:** Төв мэдээллийн систем болж, бүх агентууд үүгээр дамжин харилцана
- **Агентууд:** Зохион байгуулагч, хөрөнгө оруулалтын, мэдээний, PyFlink төлөвлөгч агентууд
- **PostgreSQL 16:** Хэрэглэгчид, хувьцааны мэдээлэл, арилжааны түүх хадгална
- **Redis 7:** Session болон response caching-д ашиглагдана

4.1.1 Агентуудын үүрэг

Хүснэгт 4.1: Хэрэгжүүлсэн агентуудын үүрэг

Агент	**Гол үүрэг**	**Технологийн чадвар**
Orchestrator Agent	Зохион байгуулагч	Gemini суурь загвараар хүсэлтүүдийг ангиллаж, агентууд руу даалгавар шилжүүлж, хариу нэгтгэх
Knowledge Agent	Зохион байгуулагчийн санах ой (Бүх агентийн мэдлэг агуулагдана)	RAG систем, vector search, зэргээр хайж дүрмийн дагуу үр дүнгээ нэгтгэнэ
Investment Agent	Шинжилгээ	МХБ өгөгдөл дүн шинжилж, хөрөнгө оруулалтын зөвлөмж өгөх
News Agent	Мэдээтэй холбоотой үйл ажиллагааг дэмжих	Finnhub API-ээр гадаад мэдээ цуглуулж хураангуйлж утга зүйн анализ хийж дамжуулах
PyFlink Planner	Төлөвлөгч	Үндсэн Kafka consumer/producer-ийг рекурсив эсвэл жирийн чиглүүлэг хийхэд тусална

4.1.2 Kafka сэдвүүд

Систем нь 12 Kafka сэдэв ашигладаг. user.requests нь хэрэглэгчийн асуултууд зохион байгуулагч руу илгээгдэх сэдэв юм. user.events нь хэрэглэгчийн бүртгэл, нэвтрэлт, profile өөрчлөлт зэрэг үйл явдлууд бичигдэнэ. agent.tasks нь зохион байгуулагчаас мэргэшсэн агентууд руу даалгавар илгээх сэдэв юм. agent.responses нь агентуудын хариулт буцах сэдэв юм. knowledge.queries нь RAG системд асуулт илгээх сэдэв юм. knowledge.results нь RAG системийн үр дүн буцах сэдэв юм. monitoring.events нь системийн мониторинг, лог бичлэг хийх сэдэв юм. planning.tasks нь PyFlink Planner руу нарийн төвөгтэй даалгавар илгээх сэдэв юм.

4.2 Технологийн стек

Системийн хөгжүүлэлтэд орчин үеийн веб технологиудыг ашиглана.

4.2.1 Frontend Технологи

Хүснэгт 4.2: Frontend технологийн стек

Технологи	**Хэрэглээ**
Nextjs 16	React-д суурилсан фреймворк, App Router, server-side rendering
React 19	UI сангийн үндэс, component-based архитектур
TypeScript 5	Static type checking, compile-time алдаа илрүүлэлт
Tailwind CSS	Utility-first CSS, хурдан responsive дизайн
Shadcn/ui	Radix UI дээр суурилсан accessible компонентууд

4.2.2 Backend Технологи

Хүснэгт 4.3: Backend технологийн стек

Технологи	**Хэрэглээ**
Node.js 20	API Gateway болон агентуудын runtime орчин
Express.js 4.18	RESTful API фреймворк, middleware систем
TypeScript 5	Backend кодын type safety хангах
Python 3.10	PyFlink Planner агентын хэл
Apache Kafka 3.5	EDA, message broker
Zookeeper 3.8	Kafka кластерийн байршуулалт
PostgreSQL 16	Үндсэн өгөгдлийн сан
Redis 7	Өгөгдөл хадгалах сав

4.2.3 Хиймэл оюуны технологи

Хүснэгт 4.4: Хиймэл оюуны технологи

Технологи	**Хэрэглээ**
Google Gemini 2.0, 2.5 Хүсэлтийг ангилах, генерац хийх, хураангуйн боловсруулах	
Sentence-Transformers	Text embedding (all-MiniLM-L6-v2 загвар, 384-dim vectors)
Finnhub API	Олон улсын зах зээлийн мэдээний эх сурвалж
TradingView Widgets	Хувьцааны тоон үзүүлэлт авах

4.2.4 Байршуулалт

Docker болон Docker Compose нь бүх сервисийг контейнержуулахад ашиглагдсан. Агент бүр өөрийн Docker контейнерт ажиллаж, бие даан өргөжих боломжтой. Vercel Hobby Plan ашиглаж демог нэвтрүүлсэн. Хаяг: https://stock-tracker-app-topaz.vercel.app/

4.3 Системийн ажиллагааны жишээ

Энэ хэсэгт системийн гол функцүүд хэрхэн ажилладагийг тайлбарлана.

4.3.1 Хувьцааны дүн шинжилгээ

Хэрэглэгч AI чат бот руу ”APU хувьцааны сүүлийн үнэ, арилжааны хэмжээг харуулаад ирээдүйд өсөх магадлал байгаа юу?” гэж асуулт асуухад дараах үйл явц өрнөнө.

Зураг 4.1-т үзүүлсэн архитектурын дагуу API Gateway нь хүсэлтийг хүлээн авч, Kafka-ийн user.requests сэдэв рүү илгээнэ. Зохион байгуулагч агент нь Gemini AI ашиглан хүсэлтийг ангилж, хөрөнгө оруулалтын агент руу agent.tasks сэдвээр даалгавар илгээнэ. Хөрөнгө оруулалтын агент нь PostgreSQL-ээс APU хувьцааны мэдээллийг авч, Gemini AI-аар дүн шинжилгээтэй хариулт үүсгэнэ. Хариулт нь agent.responses сэдвээр буцаж, API Gateway SSE холболтоор фронтенд руу дамжуулна.

4.3.2 Өдөр тутмын мэдээ илгээх

Зураг А.4-т үзүүлснээр хэрэглэгч өдөр бүр өөрийн сонирхож буй хувьцааны мэдээг э-мэйлээр авах боломжтой. Систем нь хэрэглэгчийн watchlist-ийн хувьцаануудтай холбоотой мэдээг Finnhub API-аас татаж, Gemini AI-аар хураангуйлж, утга зүйн шинжилгээ хийсний дараа э-мэйлээр илгээнэ.

4.4 Хөгжүүлэлтийн явц

Системийн хэрэгжилт ойролцоогоор 70% хийгдсэн. Бүрэн хэрэгжсэн хэсгүүд нь Docker, Kafka, PostgreSQL, Redis-ийн суурилуулалт ба агентууд (зохион байгуулагч, хөрөнгө оруулалтын, мэдээний, flink төлөвлөгч), API Gateway, Фронт дээр https://stock-tracker-app-topaz.vercel.app/ хаягаар байршуулсан байна.

4.4.1 Туршилтын үр дүн

Гүйцэтгэлийн хувьд датабаазаас мэдээлэл авахад 50-100ms, Kafka мессежийн хугацаа 5-10ms, API Gateway endpoints 200-500ms, суурь загвар ашиглан өгөгдөл гаргахад 10-20 секунд, нийт AI ашиглаж дүн шинжилгээний процесс ойролцоогоор 10-17 секунд байна.

Одоогоор EDA дээрх шинжилгээний API (Kafka → Зохион байгуулагч агент → Хөрөнгө оруулалтын агент → харуй) бүрэн ажиллаж байна. SSE streaming болон polling-аас үр дүнгээ харж байгаа. Monitoring API дээр бүх агент идэвхтэй гэсэн төлөвлөлт харуулж байна.

Өдөр тутмын мэдээний эмэйл илгээх, бүртгэлийн эмэйл илгээх зэрэг хиймэл оюуны тусламжтай агентууд амжилттай ажиллаж байна.

Дүгнэлт

Энэхүү судалгааны ажлаар хиймэл оюун агентуудыг микросервис архитектурт нэвтрүүлэх боломжийг судалж, шийдлийн зохиомжийг санал болголоо. Судалгааны явцад дараах гол үр дүнд хүрсэн:

Онолын хувьд:

Хиймэл оюуны инженерчлэл нь програм хангамж хөгжүүлэлтийн шинэ салбар болж хөгжиж байгаа нь тодорхой. Суурь загвар гарч ирэх нь аппликейшн хөгжүүлэлтийн саад бэрхшээлийг эрс багасгасан. Хэл загвараас суурь загвар хүртлээ хөгжих үйл явц нь арван жилийн технологийн дэвшлийн үр дүн юм. Өөрийгөө удирдсан сургалт, Transformer архитектур, дараах сургалт аргууд зэрэг гол түлхүүрүүд нь өнөөгийн хүчирхэг хиймэл оюун системүүдийн суурь болгосон.

Хиймэл оюун агентуудын онол нь орчин, үйлдэл, даалгавар гэсэн гурван гол бүрэлдэхүүн дээр суурилдаг. Агентууд нь төлөвлөлт, хэрэглүүрийн хэрэглээ, эргэцүүлэн бодох чадвартайгаар уламжлалт програмуудаас давуу талтай. RAG систем нь агентуудын мэдлэгийг өргөтгөж, илүү найдвартай, бодит мэдээлэл дээр суурилсан хариулт өгөх боломжийг олгодог.

Практикийн хувьд:

Микросервис архитектур дахь сервис хоорондын нарийн төвөгтэй логик, өгөгдлийн нэгтгэл, динамик routing зэрэг асуудлуудыг хиймэл оюун агентууд ашиглан шийдэж болох нь тодорхой болсон. Санал болгосон Orchestrator Agent, Service Agent, Knowledge Agent, Monitoring Agent зэрэг бүрэлдэхүүн хэсгүүд нь уян хатан, өргөжүүлэх боломжтой системийг бий болгодог.

Гэхдээ агентуудыг болит түвшинд өргөжүүлэхийн тулд зөвхөн агентын ур чадвар биш, тэдгээрийг холбодог архитектур чухал гэдгийг ойлгосон. Монолит болон gRPC/API суурилсан холболт нь монолит архитектурт тулгарсан асуудалтай адил саад болдог.

Үүнийг шийдэхийн тулд үйл явдлаар EDA ашиглан агентуудыг салангид микросервис болгон хөгжүүлэх шаардлагатай.

Микросервис архитектурт хиймэл оюун агентууд нэвтрүүлэх нь системийг илүү ухаалаг, уян хатан, хэрэглэгчид ээлтэй болгох боломжийг олгоно. Хэдийгээр одоогоор хязгаарлалтууд(Жич:хоцрогдол, өртөг, найдвартай байдал) байгаа ч технологи хурдацтай хөгжиж байгаа тул ойрын ирээдүйд эдгээр асуудлууд шийдэгдэх болно гэж найдаж байна.

Энэхүү дипломын ажил нь хиймэл оюуны инженерчлэлийн үндсийг тавьж, микросервис архитектурт агент суурилсан шийдлийг практикт хэрхэн хэрэглэж болохыг харуулсан.

Bibliography

[1] Huyen, Chip. *AI Engineering*. O'Reilly Media, 2024.

[2] Goldman Sachs Research. "Generative AI Could Raise Global GDP by 7%", 2023. https://www.goldmansachs.com/intelligence/pages/generative-ai-could-raise-global-gdp-by-7-percent.html

[3] Vaswani, A., et al. "Attention Is All You Need". *Advances in Neural Information Processing Systems*, 2017.

[4] Gao, Y., et al. "Retrieval-Augmented Generation for Large Language Models: A Survey". *arXiv preprint arXiv:2312.10997*, 2023.

[5] Yao, S., et al. "ReAct: Synergizing Reasoning and Acting in Language Models". *ICLR*, 2023.

[6] Schick, T., et al. "Toolformer: Language Models Can Teach Themselves to Use Tools". *arXiv preprint arXiv:2302.04761*, 2023.

[7] OpenAI. "Prompt Engineering Guide", 2023. https://platform.openai.com/docs/guides/prompt-engineering

[8] Newman, Sam. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015.

[9] Fowler, Martin and Lewis, James. "Microservices: A Definition of This New Architectural Term", 2014. https://martinfowler.com/articles/microservices.html

[10] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[11] Python Software Foundation. "Python 3 Documentation". https://docs.python.org/3/

[12] Ramírez, Sebastián. "FastAPI Documentation". https://fastapi.tiangolo.com/

[13] OpenAI. "OpenAI API Reference". https://platform.openai.com/docs/api-reference

[14] Anthropic. "Claude Model Card", 2024. https://www.anthropic.com/claude

[15] Facebook AI Research. "FAISS: A Library for Efficient Similarity Search". https://github.com/facebookresearch/faiss

[16] Falconer, Sean. "AI Agents are Microservices with Brains". Medium, March 2025. https://medium.com/@seanfalconer

[17] Falconer, Sean. ”The Future of AI Agents is Event-Driven”. BigDataWire, March 2025.

[18] Polak, Adi. ”Building AI Agents with Event-Driven Microservices”. Confluent Developer Advocate, 2025.

[19] Apache Kafka Documentation. ”Apache Kafka: A Distributed Streaming Platform”. `https://kafka.apache.org/documentation/`

[20] Kreps, Jay, Narkhede, Neha, and Rao, Jun. ”Kafka: A Distributed Messaging System for Log Processing”. *Proceedings of the NetDB*, 2011.

[21] Apache Flink Documentation. ”Stateful Computations over Data Streams”. `https://flink.apache.org/`

[22] Camunda Documentation. `https://camunda.com/`

[23] Inngest Documentation. `https://www.inngest.com/`

[24] BPMN Engine Documentation. `https://workflowengine.io/features/bpmn/`

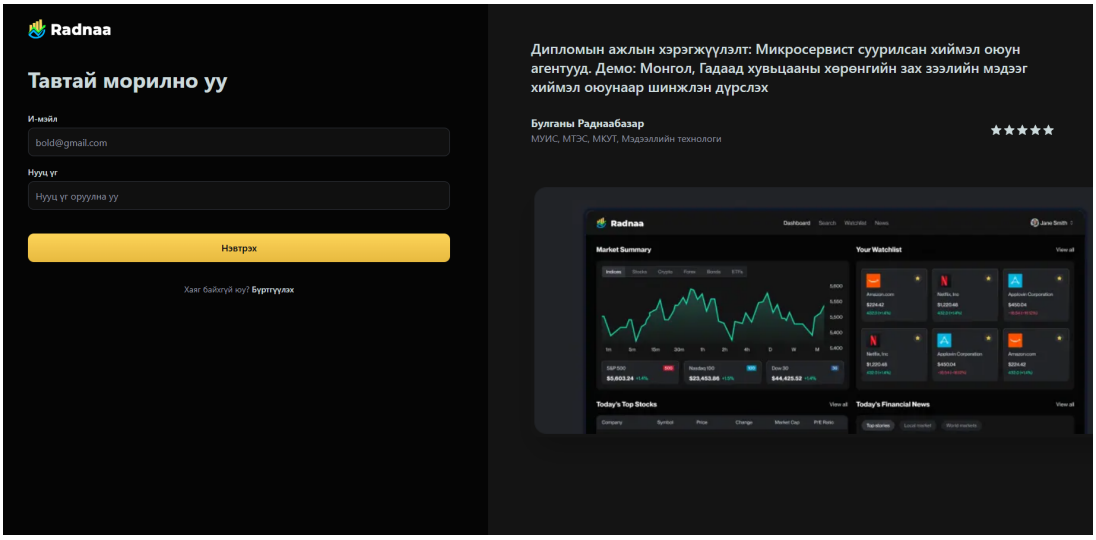
[25] Carbone, Paris, et al. ”State Management in Apache Flink: Consistent Stateful Distributed Processing”. *Proceedings of the VLDB Endowment*, 2017.

[26] Wolff, Eberhard. *Microservices: Flexible Software Architecture*. Addison-Wesley Professional, 2016.

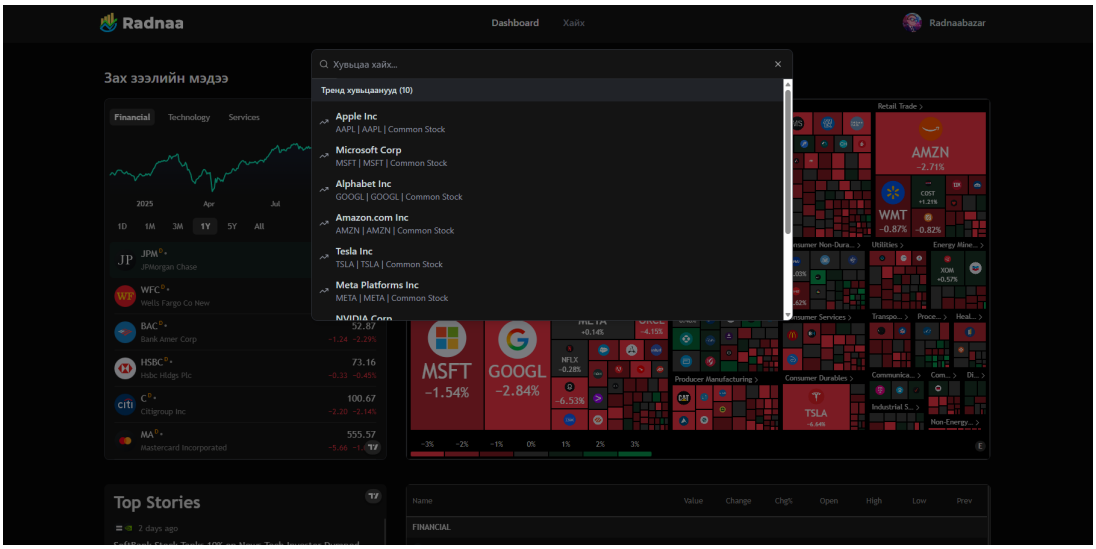
[27] Nadareishvili, Irakli, et al. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O’Reilly Media, 2016.

[28] Anthropic. ”Model Context Protocol: A Universal Standard for AI Integration”, 2024. `https://www.anthropic.com/news/model-context-protocol`

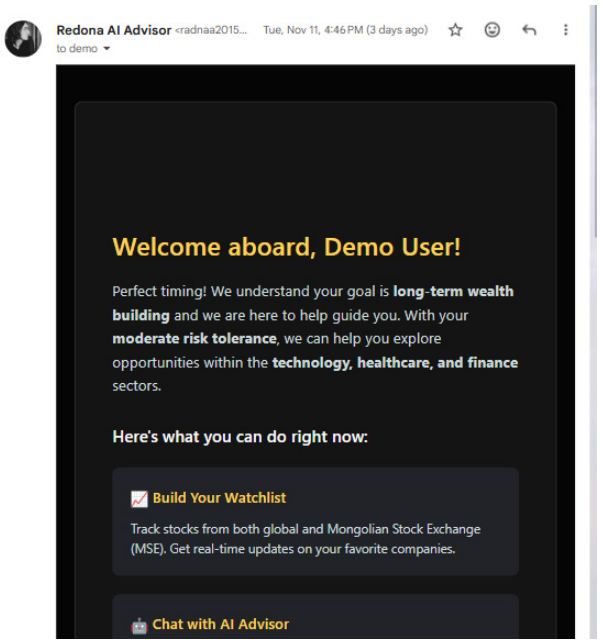
А. ВЕБ ХУУДСУУД



Зураг А.1: Нэвтрэх хуудас

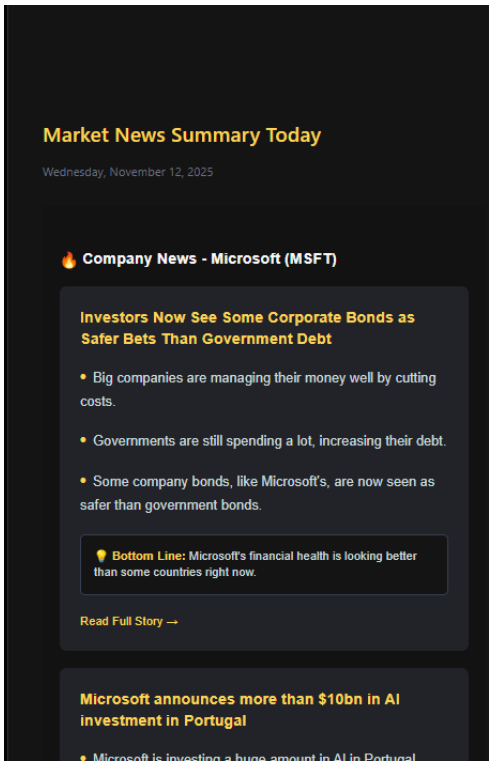


Зураг А.2: Нүүр хуудас



Зураг А.3: Бүртгэлийн мэдэгдлийг э-мэйлээр авах

Хэрэглэгч системд бүртгүүлэхэд хөрөнгө оруулах зорилго, эрсдлийн үнэлгээ, хөрөнгө оруулах зорилго зэрэг мэдээлэл оруулж байгаа ба бүртгэлийн дараа хиймэл оюун агентд уг мэдээллийг өгч, тохирсон э-мэйл агуулга гаргаж хэрэглэгч рүү илгээх



Зураг А.4: Өдөр тутмын мэдээний э-мэйл авах

Хэрэглэгч өөрийн сонирхсон хувьцааны мэдээллээ системд бүртгүүлж болох ба, хиймэл оюун ашиглаж бүх хэрэглэгчдэд өдөр тутмын сонирхсон мэдээг нь илгээж болно. FinnHub api ашиглана. Хэрвээ мэдээлэл олдохгүй бол ерөнхий мэдээ илгээнэ.

В. КОДЫН ЖИШЭЭ

В.1 Зохион байгуулагч агент: Intent Classification

Хэрэглэгчийн асуултыг ангилж, зохих агент руу чиглүүлэх Gemini AI ашигласан intent classification:

```
async classify(query: string): Promise<string> {
  try {
    // Check cache
    const cached = this.cache.get(query.toLowerCase());
    if (cached) return cached;

    // Use Gemini AI for classification
    const prompt = INTENT_CLASSIFICATION_PROMPT
      .replace('{QUERY}', query);
    const response = await geminiClient
      .generateResponse(prompt, 0.3);

    // Extract and validate intent
    const intent = response.trim().toLowerCase();
    const validIntents = ['portfolio', 'market_analysis',
      'news', 'risk_assessment', 'general_query'];
    const finalIntent = validIntents.includes(intent)
      ? intent : 'general_query';

    // Cache result
    this.cache.set(query.toLowerCase(), finalIntent);

    return finalIntent;
  } catch (error) {
    // Fallback: keyword-based classification
    return this.fallbackClassify(query);
  }
}
```

Код В.1: Intent Classification (orchestrator-agent/src/intent-classifier.ts)

В.2 Мэдлэгийн агент: RAG хайлт

Semantic search ашиглан холбогдолтой баримтуудыг олох RAG системийн хэрэгжүүлэлт:

```
function search(query: string, topK = 5, filters?: any) {
  const queryLower = query.toLowerCase();
  const keywords = queryLower.split(/\s+/)
    .filter(word => word.length > 2);

  // Score each document
  const scored = knowledgeBase.map(doc => {
    let score = 0;
    const contentLower = doc.content.toLowerCase();
    const titleLower = doc.title.toLowerCase();

    // Keyword matching
    keywords.forEach(keyword => {
```

```
      if (titleLower.includes(keyword)) score += 3;
      if (contentLower.includes(keyword)) score += 1;
    });

    // Filter matching (symbol, content type)
    if (filters?.symbol &&
        doc.metadata?.symbol === filters.symbol) {
      score += 3;
    }

    return { ...doc, score };
  });

  // Sort by score and return top K
  return scored
    .filter(doc => doc.score > 0)
    .sort((a, b) => b.score - a.score)
    .slice(0, topK);
}
```

Код B.2: RAG Search (knowledge-agent/src/index.ts)

B.3 Хөрөнгө оруулалтын агент: Event-Driven хариулт

Kafka ашиглан агентуудын асинхрон харилцаа:

```
async function handleAgentTask(message: any) {
  const { taskId, agentType, action, payload } = message;
  const startTime = Date.now();

  // Check if task is for this agent
  if (agentType !== 'investment') return;

  // Generate AI response
  const result = await generateAIResponse(action, payload);

  // Send response via Kafka
  await producer.send({
    topic: 'agent.responses',
    messages: [{
      key: taskId,
      value: JSON.stringify({
        responseId: uuidv4(),
        requestId: taskId,
        agentType: 'investment',
        status: 'success',
        result: { text: result, action },
        metadata: {
          processingTimeMs: Date.now() - startTime,
          model: 'gemini-2.0-flash'
        },
        timestamp: new Date().toISOString()
      })
    }]
  });

```

```
  // Cache response in database
  await db.query(
    `INSERT INTO agent_responses_cache
     (request_id, user_id, agent_type, query,
      response, processing_time_ms)
     VALUES ($1, $2, $3, $4, $5, $6)`,
    [taskId, payload?.userId, 'investment',
     action, result, Date.now() - startTime]
  );
}
```

Код B.3: Event-Driven Response (investment-agent/src/index.ts)

B.4 API Gateway: Server-Sent Events

Бодит цагийн агентын хариултыг хэрэглэгч рүү дамжуулах SSE streaming:

```
router.get('/stream/:requestId', async (req, res) => {
  const { requestId } = req.params;

  // Set SSE headers
  res.setHeader('Content-Type', 'text/event-stream');
  res.setHeader('Cache-Control', 'no-cache');
  res.setHeader('Connection', 'keep-alive');

  // Send initial connection
  res.write(`data: ${JSON.stringify({
    type: 'connected', requestId
  })}\n\n`);

  // Create unique consumer for this connection
  const consumer = kafkaService
    .getConsumer(`sse-${requestId}`);

  await consumer.connect();
  await consumer.subscribe({
    topic: 'agent.responses',
    fromBeginning: false
  });

  // Stream messages to client
  await consumer.run({
    eachMessage: async ({ message }) => {
      const response = JSON.parse(
        message.value.toString()
      );

      // Only send responses for this requestId
      if (response.requestId === requestId) {
        res.write(`data: ${JSON.stringify({
          type: 'response',
          data: response
        })}\n\n`);

        // Close after sending response
        await consumer.disconnect();
```

```
        res.end();
      }
    }
  });
});
```

Код B.4: SSE Streaming (api-gateway/src/routes/agent.routes.ts)