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 программ хангамж хөгжүүлэлтийн арга барилд үндсэн өөрчлөлт авчирсан. Суурь загвар гарч ирснээр программ хангамж хөгжүүлэлтийн өмнө тулгардаг саад бэрхшээл эрс буурч, шинэ боломж нээгдэж, хиймэл оюуны инженерчлэл гэсэн шинэ салбар бий болсон [1]. Goldman Sachs-ийн судалгаагаар 2025 он гэхэд АНУ-д хиймэл оюуны хөрөнгө оруулалт 100 тэрбум ам.доллар, дэлхий даяар 200 тэрбум ам.долларт хүрнэ гэж таамаглажээ [2].

Хиймэл оюуны инженерчлэл гэдэг нь бэлтгэгдсэн суурь загвар дээр программ хангамж бүтээх үйл явц юм. Машин сургалт дээр загвар бэлтгэхэд өндөр мэргэжлийн ур чадвар болон асар их өгөгдөл шаардлагатай байсан бол одоо бэлэн загварыг ашиглан аливаа программ хангамжид хиймэл оюуныг интеграц хийх боломжтой болсон нь энэхүү чиг хандлагын гол ач холбогдол юм.

Хиймэл оюун агентууд нь хиймэл оюун суурилсан программ хангамжийн хөгжүүлэлтэд онцгой боломжуудыг нээж өгч байна. Агент гэдэг нь өөрийн орчныг мэдрэх, түүн дээр үйлдэл хийх чадвартай систем юм [1]. Хиймэл оюун агентууд нь том хэлний загварын хүчийг ашиглан даалгаврууд ойлгож, төлөвлөгөө гаргаж, олон алхам бүхий үйл ажиллагааг гүйцэтгэх чадвартай. Гэвч байгууллагууд агентуудыг системгүй байршуулах нь мөшгөхөд хүндрэлтэйгээр дамжин унаж, бизнесийн үйл ажиллагаа саатаж байдаг [14]. Микросервис архитектур нь том системийг жижиг, бие даасан сервисүүдэд тархмал байдлаар хуваах замаар уян хатан, өргөтгөх боломжтой, найдвартай системүүд бий болгодог. Эдгээр агентуудыг микросервис архитектурт нэвтрүүлснээр эдгээр асуудлыг шийдэж, системийн ухаалаг төлөвлөгч, өгөгдөл боловсруулагч, ажлын урсгалын удирдагч зэрэг үүргийг гүйцэтгүүлэх боломжтой [1].

Энэхүү судалгааны ажил нь хиймэл оюун агентууд болон микросервис архитектурын уялдааг судалж, практик шийдлүүдийг санал болгохыг зорьсон. Суурь загварын онол, агентуудын төлөвлөлт, хайлтаар нэмэгдүүлсэн үүсгэлт болон эдгээрийг микросервис архитектурт хэрхэн нэгтгэх талаар авч үзсэн болно.

Нэр томьёоны тайлбар

Англи нэр	Монгол орчуулга
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	Үзэгдэлд суурилсан архитектур
Message Queue	Мессежийн дараалал
Vector Database	Векторын өгөгдлийн сан
Embedding	Векторчилсон хэсгийн төлөөлөл
Token	Токен (хэлний загвар дахь тэмдэгт мөрийн таслалын нэгж)
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	Төлөвлөгч
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 чанартай байдал
Coupling	Хамаарал
Decoupling	Тархмал байдал
Synchronous communication	Синхрон холбоо
Asynchronous communication	Асинхрон холбоо
Partition	Хэсэглэл
Topic	Сэдэв

Товчилсон үгс

Товчлол	Нэр томьёо	Монгол тайлбар
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	Үзэгдэлд суурилсан 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

Нэр томъёо	Тайлбар
Docker	Контейнержуулалтын платформ; сервис бүрийг тусдаа орчинд ажиллуулах боломж олгодог.
Apache Kafka	Тархмал 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	Тархмал 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, Үзэгдэлд суурилсан архитектур (EDA) ашиглан уян хатан, өргөжих боломжтой системийн зохиомж гаргах юм. Уг зохиомжийн үр ашигтай байдлыг бататгахын тулд Монголын хөрөнгийн биржийн бодит өгөгдөлд суурилсан демо систем хөгжүүлж туршиж үзэх болно.

Зорилт:

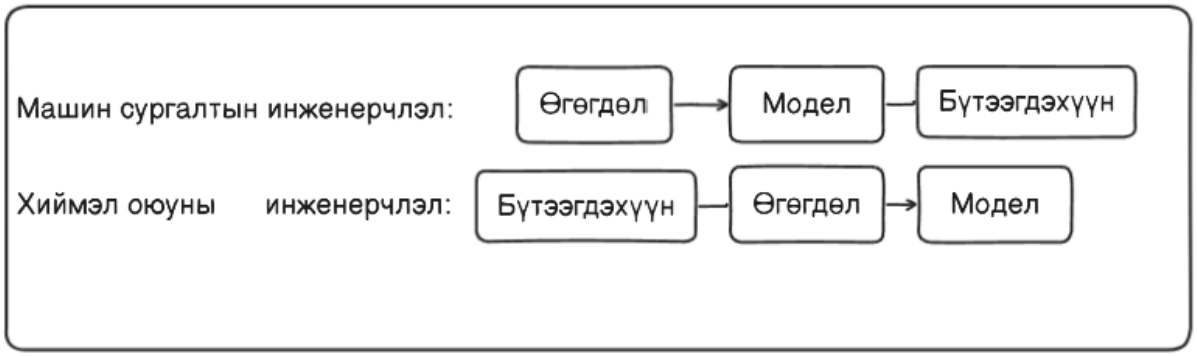
1. Хиймэл оюуны инженерчлэл, суурь загвар, RAG, агентуудын онол, онцлогийг судлах
2. Микросервис архитектур болон EDA-ийн давуу талыг тодорхойлох
3. Apache Kafka, Apache Flink ашиглан агентуудыг тархмал микросервист нэгтгэсэн зохиомж гаргах
4. Монголын хөрөнгийн биржийн өгөгдөлд суурилсан демо систем хөгжүүлж, санал болгосон зохиомжийг бататгах

1. ХИЙМЭЛ ОЮУНЫ ИНЖЕНЕРЧЛЭЛ БА АГЕНТУУД

Энэхүү бүлэгт хиймэл оюуны инженерчлэлийн үндсэн ойлголтууд, суурь загварын хөгжил, промпт инженерчлэл, цаашлаад RAG, олон агент системийн 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 өсөж 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 Загварын үнэлгээ

Суурь загварыг үнэлэх нь эрсдлийг бууруулах, боломжуудыг илрүүлэх тал дээр чухал ач холбогдолтой. Үнэлгээ нь загварыг сонгох, үр дүнг хэмжих, бодит хэрэглээнд нэвтэрч болох 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.

1.5 Промпт инженерчлэл

Промпт инженерчлэл гэдэг нь загвараас хүссэн үр дүнг гаргуулахын тулд промпт заавар бичих үйл явц юм [6]. Энэ нь загварын жинг өөрчлөхгүйгээр хиймэл оюуны зан чанарыг удирдах хамгийн хялбар бөгөөд түгээмэл 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. [6] 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 найдваргүй байж болдог. Түүнчлэн нарийн төвөгтэй даалгавруудыг хялбар дэд даалгавруудад хувааж өгөх нь үр дүнтэйгээр бага токен ашиглах боломжийг олгоно.

1.5.2 Промптын хамгаалалт

Программ хангамж бодит орчинд байрших үе шатанд ормогц хортой довтолгооноос хамгаалах шаардлагатай болдог. Энэхүү хортой код нь хэрэглэгчийн хувийн хэвшлийн эсвэл байгууллагын нууц мэдээллийг хиймэл оюуны цоорхойгоор авах оролдлого юм. Үүнд загварын сургалтын өгөгдөл, контекстын мэдээлэл зэрэг багтана. [1]

Иймээс хиймэл оюуны агент эсвэл загвар хөгжүүлж буй тохиолдолд процессын оролт болон гаралтыг хамгаалах шаардлагатай байдаг.

1.6 Хайлтаар нэмэгдүүлсэн үүсгэлт (RAG)

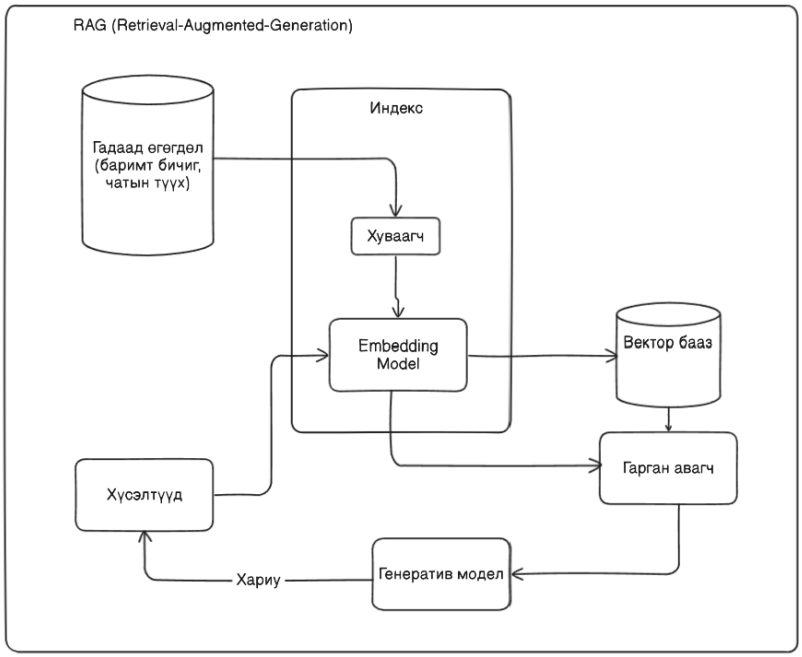
RAG буюу хайлтаар нэмэгдүүлсэн үүсгэлт нь загварын мэдлэгийг гадаад эх сурвалжаар өргөтгөх арга юм. Энэ нь загварын дотоод мэдлэг нь хангалтгүй, хуучирсан эсвэл алдаатай байх асуудлыг шийддэг [1].

Хэдийгээр загварын контекстийн урт тогтмол нэмэгдэж байгаа ч RAG-ийн ач холбогдол алдагдахгүй байна. Зарим аппликейшнд өгөгдлийн хэмжээ байнга өсөж байдаг. Иймээс RAG удааширч магадгүй тул үүнийг сайтар үнэлж, тасралтгүй хөгжүүлэлт хийснээр бодит хэрэглээнд үр нөлөө алдахгүй байх боломжийг бүрдүүлнэ. Урт контекстийг боловсруулж чаддаг гэдэг нь тэр контекстийг 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 бүрд 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-ийн бүтэц

RAG 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 [1]

Зураг 1.2-аас харахад хайгч нь асуултад хамгийн их холбоотой баримтуудыг олж авдаг бол үүсгэгч нь олж авсан баримтуудыг суурь загварт дамжуулан 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 векторт хэсэгчилж хуваагдахад өгүүлбэрүүд утга зүй бусаар хуваагдаж, хайлт хийх боломжгүй болдог. Иймээс хайлтын алгоритмаа 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 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-ыг сайжруулах аргууд

RAG-ийг 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.

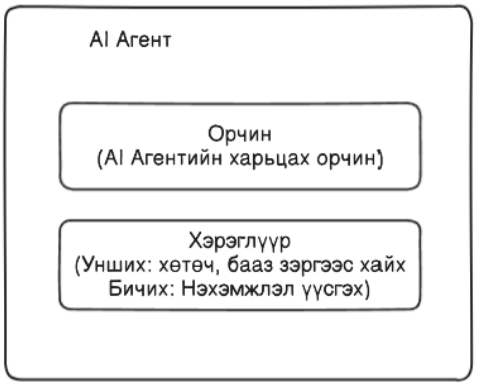
1.7 Хиймэл оюуны агентууд

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 орчин ба хэрэглүүр гэсэн хоёр гол хэсгүүд байдаг. [1]. Орчин нь аливаа агентийн ажиллах орчин ба жишээ нь интернэт, гал тогоо зэрэг байж болно. Энэхүү орчинд үйлдэл хийх боломжийг үүсгэж өгдөг зүйл бол хэрэглүүр ба хэрэглүүрүүдийг хангаснаар агентийн чадамжийг өргөсгөдөг. Үүнийг зураг 1.3-т дүрслэв.



Зураг 1.3: Агентын бүрэлдэхүүн хэсгүүд [1]

1.7.3 Хэрэглүүрүүд

Хэрэглүүр байхгүй бол агентын чадамж маш хязгаарлагдмал байх болно. Уян хатан шийдвэр гаргалт, найдвартай гүйцэтгэлийн хоорондох хоосон зайг нөхдөг онцлогтой. Хэрэглүүрийг гурван ангилалд хуваагддаг [1].

1. Мэдлэг нэмэгдүүлэх хэрэглүүрүүд нь хайлтаар нэмэгдүүлсэн үүсгэлт систем буюу RAG-ын бүрэлдэхүүн хэсгүүд юм. Үүнд текст хайгч, зураг хайгч, SQL гүйцэтгэгч, интернэтийн орчинд хайлт хийх, дотоод хайлтын системүүд зэрэг багтана.

2. Чадамж өргөтгөх хэрэглүүрүүд нь агентын үндсэн чадварыг өргөжүүлдэг. Хиймэл оюун загварууд математикт сул байдаг тул агентд бэлэн тооцоолуурын API өгснөөр тооцоог оновчтой, хурдан, токены бага зарцуулалтаар гүйцэтгэх боломжийг үүсгэдэг.

3. 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.

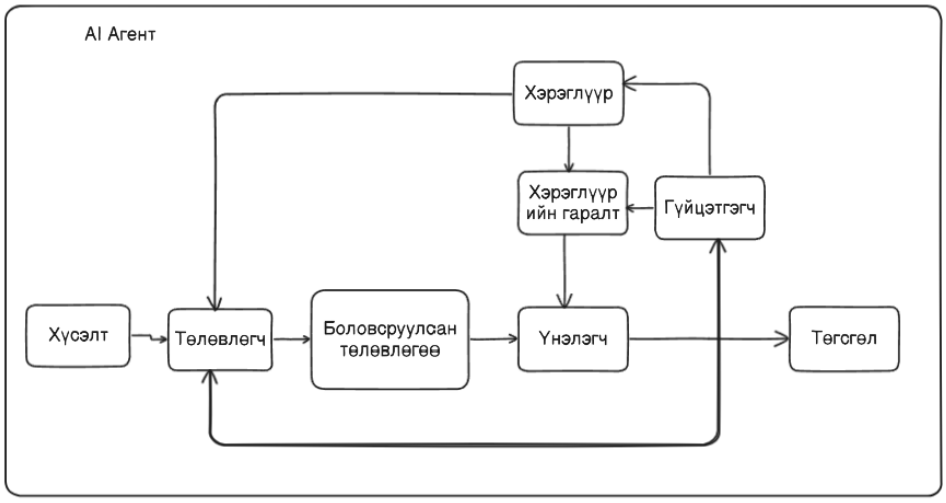
1.7.4 Агентын санах ой

Хиймэл оюуны 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 мессежнүүд 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.

1.8 Төлөвлөгч агент (ReAct agent)

ReAct (Reasoning Acting) агент буюу төлөвлөгч агент нь хэрэглүүрийн тусламжтайгаар аливаа асуудлыг эргэцүүлэн бодож шийдэл гаргадаг фреймворк юм [5] [1]. Үндсэндээ ямарваа нэгэн орчинд олон агентуудыг уяж ажлуулдгаараа онцлогтой. Энэ агент нь даалгаврыг алхам алхмаар бодож задлаад, шаардлагатай үед хайлт, тооцоолуур, API зэрэг гаднын хэрэглүүрүүдийг дуудах замаар шийдэл гаргаж ажилладаг. Ингэснээр уламжлалт чатботоос илүү уян хатан, дасан зохицох, асуудлыг шийдвэрлэх чадвартай байдаг [5]. ReAct нь RAG-ийн хослолоор алдаа багатай, баримтын үндэслэлтэй хариу гаргах давуу талтай.

Зураг 1.4-т эхний үе шат нь хүсэлтийн дагуу төлөвлөгөө үүсгэх юм. Төлөвлөгч нь орчны мэдээллийг аваад 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 дахин сайжруулж болох лог үлдээнэ.



Зураг 1.4: Агентын төлөвлөлт [1]

1.8.1 Яагаад олон агентын зохиомж хэрэгтэй вэ?

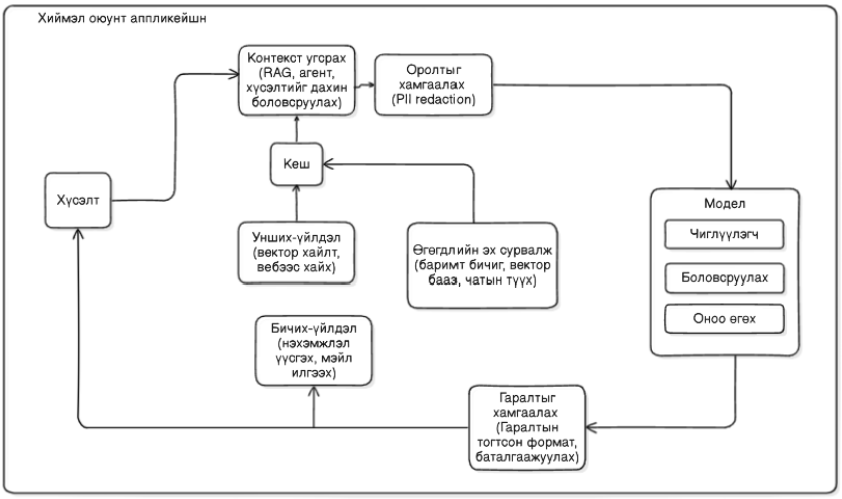
Агент нь тодорхой даалгаврын хүрээнд л процесс гүйцэтгэх чадвартай байдаг. Жинхэнэ бизнесийн үйл ажиллагаа нь олон дэд процессд хуваагдаж болдог шиг агентийн хэрэглээ мөн адил олон хуваагдаж болдог. Тиймээс олон агентийн зохиомж нь ялгаатай орчинд нарийн төвөгтэй асуудлыг шийдэх, дасан зохицох, үр дүнтэй 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.

1.8.2 Суурь загварууд төлөвлөгч болж чадах уу

Зарим судлаачид суурь загвар нь төлөвлөгч болж чадахгүй гэж үздэг. [13]. Учир нь төлөвлөлт нь үндсэндээ хайлтын асуудал бөгөөд авторегрессив буюу магадлалт суурилсан загвар нь үргэлж ялгаатай, буруу эсвэл төөрөгдсөн хариулт өгдөг. Гэвч бодит байдал дээр загвар өөр өөрийнхөө үүсгэсэн төлөвлөгөөг үнэлээд дахин эхэлж чаддаг тул сургалт сайн хийснээр төөрөгдлийг багасгаснаар 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. Функцуудийг 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].

1.9 Жишээ хиймэл оюунт программ хангамжийн зохиомж



Зураг 1.5: Хиймэл оюунт аппликейшн [1] [13]

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.

1.10 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 [19] [18].

Inngest нь serverless workflow платформ бөгөөд үзэгдэлд суурилсан архитектур ба хэрэглэгчийн функцийн түвшинд ажилладаг. LLM интеграци байдаг ч RAG систем дутмаг байна. Систем нь асинхрон харилцаа, параллель боловсруулт дэмждэгээрээ

найдвартай байдаг. Уг систем нь үүлэн технологи дээр байршдаг ба хаалттай эхийн программ хангамж юм. Голдуу startup, жижиг багуудад зориулагдсан.

Temporal нь хиймэл оюуны агенттай дэмжлэгтэй ба үйл явдлын төлөв байдлаа хадгалж, унасан ч яг өмнөх төлөв дээр буцан сэргэж чаддаг найдвартай ажиллагаатай платформ юм. Workflow хэлбэрээр л процесс гүйцэтгэгдэх тул өргөжихөд хүндрэлтэй байдаг. Мөн хаалттай эхийн программ хангамж юм. Uber, Netflix, Stripe зэрэг томоохон компаниуд ашигладаг.

Дээрх системүүдээс ялгаатай нь энэ судалгааны санал болгож буй зохиомж нь хиймэл оюун агентуудыг анхнаасаа бие дан өргөжих тархмал микросервис болгон хөгжүүлж, үйл явдлыг лог хэлбэрээр хадгалж найдвартай байдлыг хангахаар зохиомжлогдсон. Энэ нь бодит цагийн шийдвэр гаргалт, аудит тест хийх боломжийг нэмэгдүүлэх ба шинэ агентийг хялбараар нэмэх давуу талтай. Мөн энэ ажилд төлөвлөгч агент байх ба энэ нь динамикаар асуудлыг шинжилж, зөв хэрэглүүр дуудаж, процессийг гүйцэтгэж чаддагаар онцлогтой [5]. Kafka-Flink зэрэг нээлттэй эхийн технологи нэвтрүүлж лог хадгалах нь Temporal шиг дахин сэргэх чадвар боломжийг нээж, логуудыг үнэлээд уг программ хангамжийг сайжруулах боломжийг олж нээж болно. Гэвч, найдвартай төлөвлөгч агент болон олон агентыг уялдуулж ажиллах нь хөгжүүлэлтийн хувьд нөөц их шаардана. Энэхүү зохиомж нь олон агентийг хямд, найдвартай, системтэйгээр нэвтрүүлэхийг зорьж буй байгууллагуудад зохимжтой.

1.11 Бүлгийн дүгнэлт

Энэхүү бүлэгт хиймэл оюуны инженерчлэлийн үндсэн ойлголтуудыг тайлбарлалаа. Хиймэл оюуны инженерчлэл нь бэлтгэгдсэн суурь загвар дээр программ 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 сургалтаас авсан 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 зэрэг одоо байгаа системүүдтэй харьцуулахад энэ судалгааны санал болгож буй зохиомж нь хиймэл оюун агентуудад зориулан тусгайлан бүтээгдсэн, Kafka-Flink суурилсан, нээлттэй эхийн технологи ашигладгаараа онцлогтой.

2. МИКРОСЕРВИС БА АГЕНТУУД

Энэхүү бүлэгт микросервис архитектурын онолыг судлах ба үзэгдэлд суурилсан 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. [20]

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 [20]. Энэ өөрчлөлт нь багуудад тархмал 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 программ 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. [20].

Нэгт, бие даан ажиллах чадвартай байдаг. Микросервис бүр ялгаатай бизнесийн логик агуулах ба шаардлагатай бол өөрийн өгөгдлийн сантай байдаг. Энэ нь сервис бүр бусад сервисээс хараат бус ажиллах боломжийг олгодог. Хоёрт, уян хатан хөгжүүлэлэх боломж юм. Өөр өөр багууд өөр өөр технологи, програмчлалын хэл ашиглан өөрсдийн сервисийг хөгжүүлж болдог. Энэ нь багууд өөрсдийн мэргэжлийн чиглэлд тохирсон технологи сонгох эрх чөлөөг өгдөг. Гуравт, өргөжих чадвар юм. Системийг нэгтгэсэн байдлаар өргөжүүлэх шаардлагагүй болно. Дөрөвт, найдвартай байдал ба хэрэв нэг сервис алдаа гарч унавал бусад сервисүүд хэвийн ажиллаж үргэлжлэнэ. Энэ нь системийн тогтвортой байдлыг хангадаг.

2.2 Микросервисийн давуу тал

Микросервис архитектур нь монолит системээс олон талаараа давуу талтай байдаг. Технологийн 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. [21].

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 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. [21].

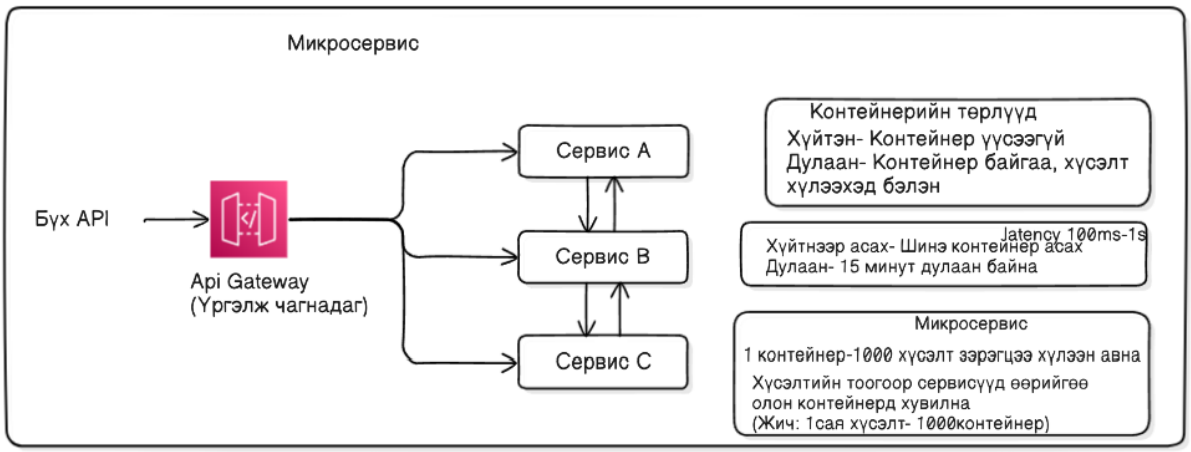
Сүлжээний хоцрогдлын талаас авч үзвэл сервисүүд хоорондоо сүлжээгээр харилцдаг учир нэмэлт хоцрогдол гардаг бөгөөд энэ нь системийн ерөнхий гүйцэтгэлд нөлөөлдөг. Олон сервисүүдээр дамжин явах хүсэлтийн алдааг олж тодорхойлох, засах ажил үйл ажиллагаа төвөгтэй байдаг. Сервисүүд хэрхэн үр дүнтэй харилцах, ямар протокол ашиглах, өгөгдлийн формат хэрхэн нийцүүлэх зэрэг олон нарийн асуудлыг шийдэхийг шаарддаг. Эцэст нь өгөгдлийн сангийн эсвэл үйлдлийн системийн транзакцийн удирдлагын асуудал нь олон сервисүүдээр транзакци явуулах нь өгөгдлийн сангууд түгжигдэх, тогтворгүй байдал үүсэх эрсдлийг нэмэгдүүлдэг.

2.4 Микросервис хоорондын харилцаа

Микросервисүүд хоорондоо дараах хоёр аргаар харилцдаг: [21].

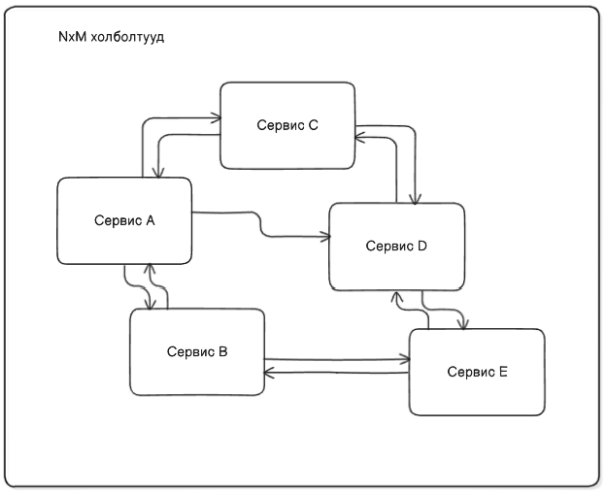
2.4.1 Синхрон харилцаа

Синхрон харилцаа нь HTTP REST API эсвэл gRPC ашиглан шууд хүсэлт илгээж хариу хүлээдэг арга юм. Энэ арга нь хэрэгжүүлэхэд харьцангуй энгийн боловч сул талтай. Нэгт, нягт хамаарал буюу tight coupling үүсгэдэг. Сервисүүд бие биенээсээ шууд хамаарч байдаг учир нэг сервис өөрчлөгдөх үед бусад сервист нөлөөлдөг. Хоёрт, нэг сервис унавал түүнээс хамааралтай бусад сервисүүд ч гэсэн дамжин унадаг. Гуравт, хоцрогдол нэмэгддэг. Сервисүүд бие биенээсээ цуваа байдлаар хариу хүлээж байдаг учир хариул авах цаг удаан байх тусам ерөнхий системийн хоцрогдол нэмэгддэг.



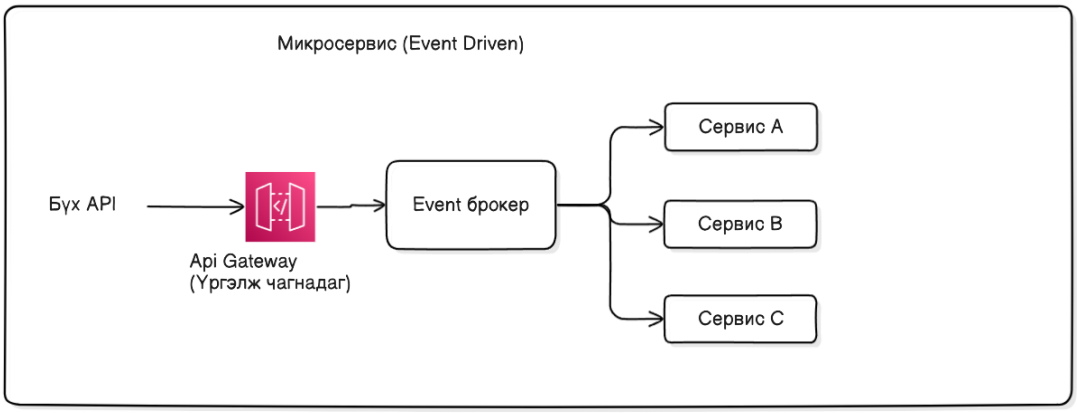
Зураг 2.1: Синхрон микросервис [23]

Сервисийн тоогоор NxM холболтын харьцаагаар холболт нэмэгдэх учир үүнийг найдвартай удирдахад маш хүндрэлтэй болно. Зураг 2.2-д уг холбоосуудыг дүрсэлсэн ба бие биетэй хамааралтай байхад дамжин унах магадлал авчирдаг. Энэ ч утгаараа олон агент нэвтэрвэл дамжин унах эрсдэлтэй.



Зураг 2.2: Синхрон микросервисийн NxM холбоо [14]

2.4.2 Асинхрон харилцаа



Зураг 2.3: Асинхрон микросервис [23]

Асинхрон харилцаа нь мессежийн дараалал ашигладаг арга бөгөөд RabbitMQ, Apache Kafka зэрэг технологиудыг ашиглан мессеж солилцдог. Энэхүү аргууд дараах онцлогтой. Эхний онцлог нь тархмал байдлаар ажиллах боломжийг бүрдүүлдэг. Сервисүүд 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 боловсруулах боломжтой болдог. Гурав дахь онцлог нь синхрон харилцаанаас илүү төвөгтэй хэрэгжилттэй байдаг. Мессежийн формат тодорхойлох,

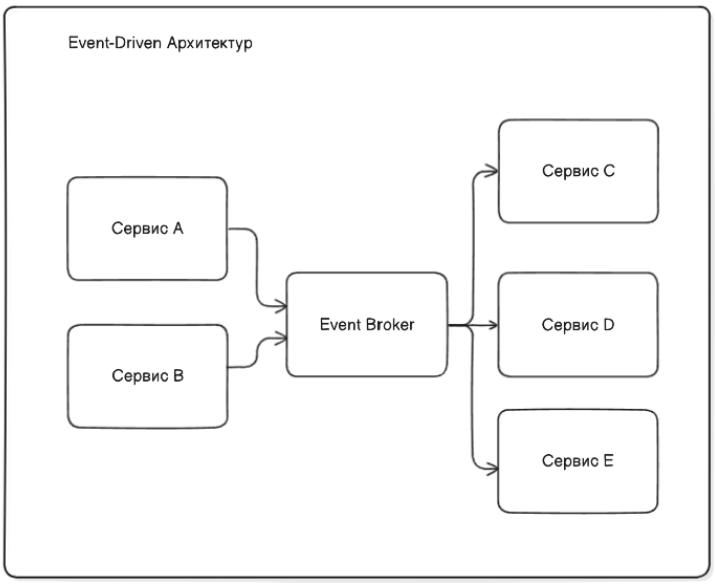
алдааны менежмент хийх, мессежийн дараалал хангах зэрэг нэмэлт асуудлыг шийдэх шаардлагатай.

2.5 Үзэгдэлд суурилсан архитектур буюу EDA

2.5.1 EDA гэж юу вэ

Микросервисүүд гарч ирснээр шинэ сорилт бий болсон [21]. Хэрэв сервисүүдийг шууд gRPC эсвэл API-ээр холбовол асар том хамаарлын сүлжээ үүсдэг. Хэрэв нэг сервис унавал энэ нь холбогдсон замын дагуух бүх мөчир буюу node-д нөлөөлнө. Мөн бие биенийгээ цуваа байдлаар хүлээснээр гацалт маш ихээр үүсдэг.

EDA нь энэхүү асуудлыг шийдэж болдог. Нягт холбогдсон, синхрон харилцааны оронд EDA нь бүрэлдэхүүн хэсгүүдэд event-driven асинхрон харилцах боломжийг олгодог. Сервисүүд бие биенийгээ хүлээхгүйгээр, параллель боловсруулалт хийдэг. Зураг 2.4-д NxM сүлжээний хамаарлыг арилгаж чадсан байна.



Зураг 2.4: EDA суурилсан микросервис [14]

2.5.2 EDA-ын давуу тал

Сервисүүд үйл явдлаар (event) харилцдаг учир тэд бие биенээсээ бүрэн хараат бус байж тархмал шинжтэй болдог. Нэг сервис өөрчлөгдөх эсвэл түр зуур унах нь

бусад сервист шууд нөлөөлөхгүй бөгөөд энэ нь системийн уян хатан чанартай байдлыг нэмэгдүүлдэг.

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 бусад сервисүүдийн ажиллагаа хэвээр байдаг. 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 тархмал, өндөр дамжуулалттай, бага хоцрогдолтой EDA-ийн төв хэсэг болж чаддаг. [16].

Kafka-гийн үндсэн ойлголтууд

Apache Kafka-гийн архитектур нь таван үндсэн ойлголттой [16]. Сэдэв буюу topic нь үйл явдлуудыг (event) ангилах логик сувгийн 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.

Хэвтээ өргөжих чадварын хувьд Kafka-гийн тархмал зохиомж нь шинэ сервис, эсвэл агент нэмэх боломжийг олгодог. Системийн ачаалал нэмэгдэх тусам partition нэмж, сервис нэмж өргөжүүлэх нь маш энгийн. Эцэст нь дахин ачааллуулж гүйцэтгэх боломжийн хувьд Kafka нь тархмал лог ашигладаг учраас үйл явдал бүр хадгалагдаж, алдаа засах, үнэлгээ хийх, загварыг дахин сургахад тохиромжтой.

2.5.4 Apache Flink

Apache Flink нь төлөв байдал удирддаг үйл явдлын боловсруулалт хийх чадвартай тархмал процессын удирдлагын фреймворк юм. [17] Flink нь Kafka-тай хамт ашиглах замаар хүчирхэг бодит цагийн өгөгдөл боловсруулалтын систем бүтээх боломжийг олгодог.

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 хүрээний өгөгдлийг ашиглан нарийн шинжилгээ хийж чаддаг. Үйл явдлын цаг дээр үндэслэн бодит цагийн шинжилгээ хийх чадвартай. Энэ нь хоцрогдсон мессежнүүдийг 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” эшлэлээс [17] авч үзвэл мэдээлэл яг нэг удаа л боловсруулагддаг. Энэ нь өгөгдөл алдагдах эсвэл давхардах асуудлыг шийддэг.

2.6 Flink ба ReAct төлөвлөгч

Flink нь том хэлний загвартай ажиллах чадвартай. [14] Flink нь өгөгдлийг процессын төлөв байдалд хадгалж, том хэлний загвар руу илгээж, хариу аваад төлөв байдалдаа олон мэдээлэл нэгтгэх боломжийг олгодог. Энэ нь төлөвлөгч агентыг Flink болгон хөгжүүлэх зохимжтой байдаг [15].

Жишээлбэл, эхлээд Kafka сэдвээс үйл явдлыг (events) уншина. Дараа нь аливаа нэгэн загвар ашиглан цар хүрээг ойлгох, даалгаврыг задлах, төлөвлөгөө гаргах үйл явцыг гүйцэтгэнэ. Үр дүнг өөр Kafka topic руу бичнэ. Эцэст нь энэ topic-ийг хүлээж авах агентууд энэ үр дүнг уншиж процессоо гүйцэтгэнэ. Иймээс ReAct(Reasoning Acting) буюу төлөвлөлтийн дагуу ажиллаж чаддаг байна. Үүнийг хэрэгжүүлэлт дээр бататгах болно.

2.7 Бүлгийн дүгнэлт

Энэхүү бүлэгт микросервис архитектурын онол, практик, түүнчлэн EDA-ийн давуу талыг дэлгэрүүлэн судалсан. Монолит системээс микросервис рүү шилжих нь программ хангамжийн хөгжлийн чухал дэвшил болов. [14]

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 тархмал 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 (kafka event) асинхрон харилцах боломжийг олгодог. Тархмал 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. [15] Kafka-ийн хэвтээ өргөжих чадвар, бага хоцрогдол, тархмал 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 тархмал 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 тархмал 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. EDA, 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. ШИЙДЭЛ БА ЗОХИОМЖ

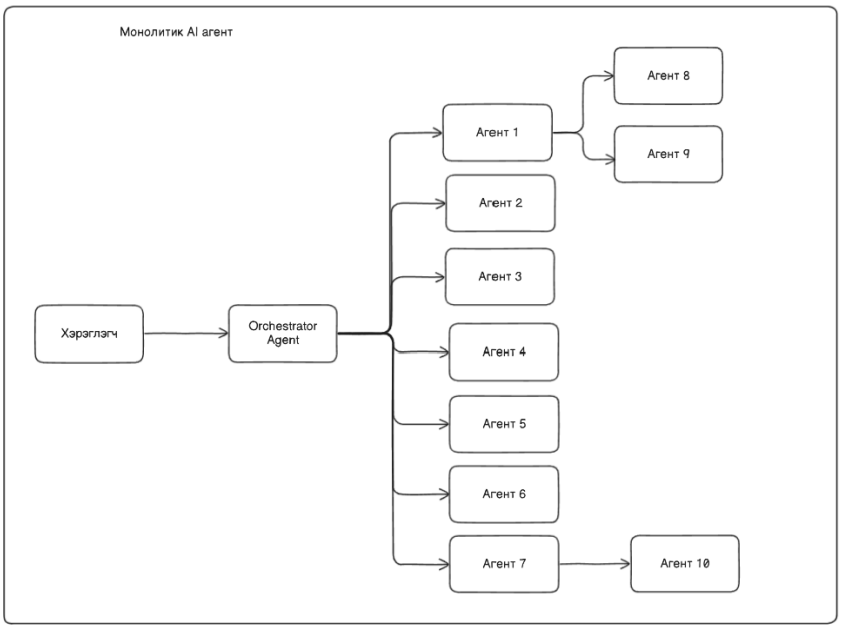
Энэхүү бүлэгт олон агентын системийг монолит зохиомжоор хөгжүүлэх эрсдэл ба асуудлуудыг тодорхойлно. Дараа нь тархмал байдлаар агентуудыг микросервис болгон хөгжүүлэх зохиомжийн шийдлийг санал болгоно.

3.1 Монолит агентуудын эрсдэл

Хиймэл оюун хөгжүүлэхэд тогтсон стандарттай фреймворк байдаггүй. [13]. 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 [18]. 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. [13]. 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.



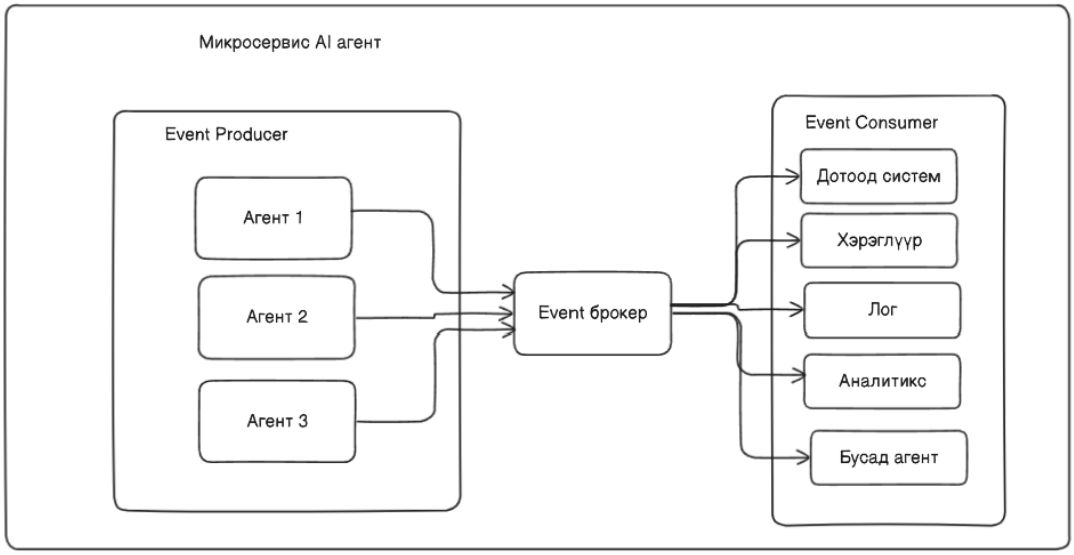
Зураг 3.1: Монолит агентын архитектур: NxM нягт холбоос [13] [18]

Зураг 3.1-аас харахад монолит 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 тархмал болгодог.



Зураг 3.2: EDA орчинд агент хоорондын харилцаа [14]

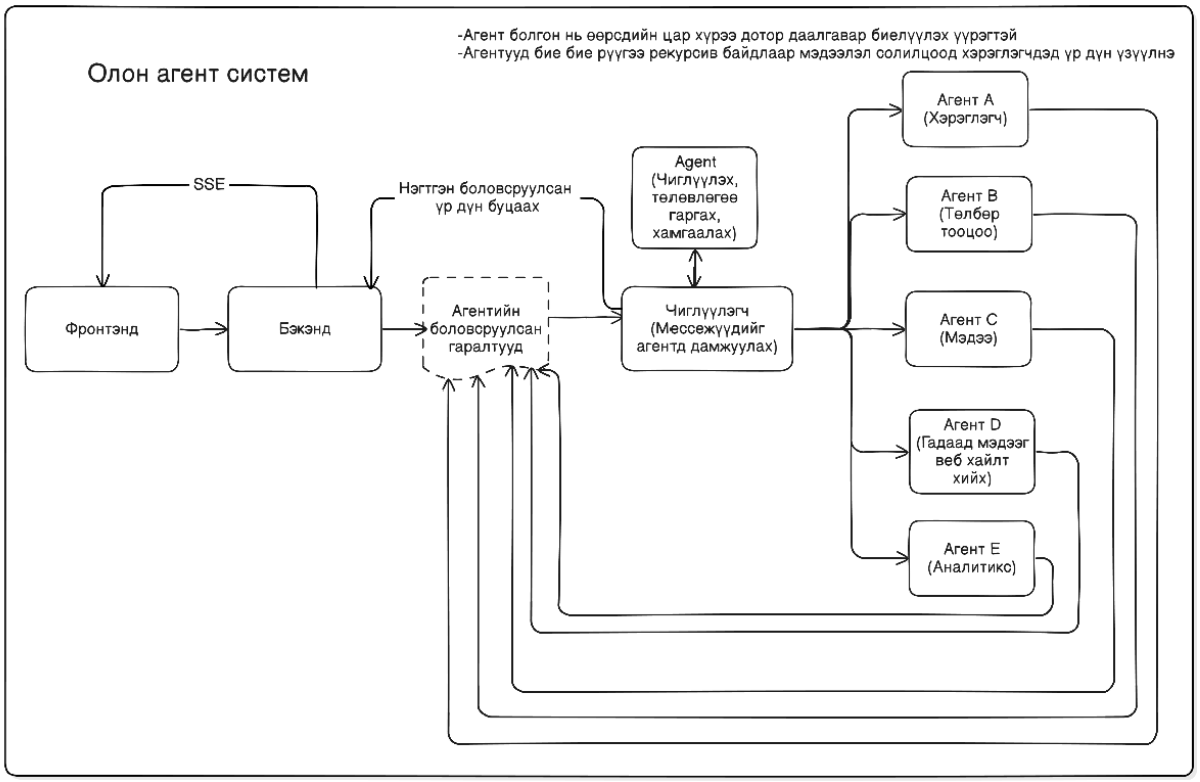
Зураг 3.2-т үйл олон агентын системийн агент хоорондын харилцааг дүрсэлсэн. Зураг 3.2-аас харахад систем нь гурван гол бүрэлдэхүүнтэй:

- Эвент үүсгэгчүүд: Өөр өөр зориулалттай агентууд (төлөвлөгч, чиглүүлэгч, мэдээ, төлбөр тооцооны гэх мэт) үйл явдал (kafka event) 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. И

Энэхүү архитектур нь монолит системийн бүх сул талыг шийддэг. Тархмал 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 тархмал 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: Микросервис архитектурд суурилсан олон агент системийн зохиомж [14] [1]

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. Төлөвлөгч агент суурь загвараар хүсэлтийг ангилж, 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 зэрэгцээ байдлаар боловсруулна
- Уян хатан өргөтгөл: Шинэ агент нэмэх нь бусад агентын кодыг өөрчлөхгүй
- Динамик шийдвэр гаргалт: Хиймэл оюун дараагийн алхмыг динамикаар төлөвлөж гүйцэтгэнэ.
- 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, EDA ашиглан тархмал, 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 зэрэг одоо байгаа системүүдтэй харьцуулахад энэхүү судалгааны санал болгож буй зохиомж нь анхнаасаа хиймэл оюун агентуудыг тархмал 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 тархмал 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 үүрэг, ажиллах зарчмыг тайлбарлаад, дараа нь хэрэглэгчийн боломжууд, өгөгдлийн урсгал, технологийн стек, туршилтын үр дүнг дэлгэрэнгүй авч үзнэ.

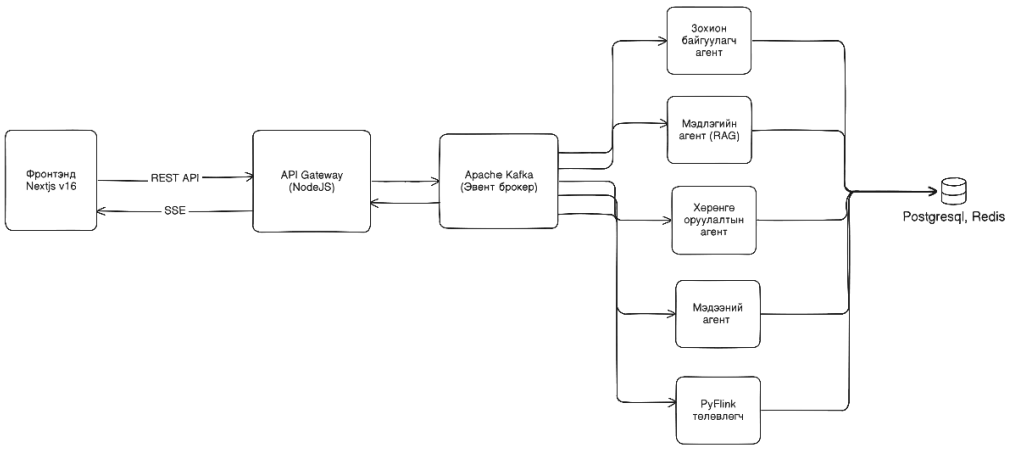
4.1 Демо системийн тойм

Демо систем нь Монголын Хөрөнгийн Биржийн (МХБ) бодит өгөгдөлд суурилсан хөрөнгө оруулалтын зөвлөгөө өгдөг веб платформ юм. Систем нь хиймэл оюун агентуудыг микросервис архитектурт EDA (Үзэгдэлд суурилсан архитектур) хэлбэрээр нэвтрүүлж, хэрэглэгчдэд хувь хүнд тохирсон монгол хэл дээрх дүн шинжилгээ, зөвлөмж өгөх боломжийг олгоно.

Системийн гол онцлогууд:

- Олон агентын архитектур: Зохион байгуулагч, хөрөнгө оруулалтын, мэдээний, мэдлэгийн, төлөвлөгч гэсэн таван агент
- Хувийн зөвлөмж: Хэрэглэгчийн профайл (эрсдлийн хүлээцтэй байдал, хөрөнгө оруулалтын зорилго) дээр суурилсан шинжилгээ
- Монгол хэлний дэмжлэг: Бүх хиймэл оюуны хариултууд монгол хэлээр
- Бодит өгөгдөл: МХБ-ийн 2015-2024 онд явагдсан арилжааны түүхэн мэдээлэл

4.2 Системийн архитектур



Зураг 4.1: Демо системийн архитектур

Зураг 4.1-т демо системийн архитектурын зохиомжийг дүрсэлсэн. Системийн гол бүрэлдэхүүнүүдийг дараах дарааллаар тайлбарлана.

4.2.1 Frontend давхарга

Next.js 16 фреймворк ашиглан React дээр суурилсан хэрэглэгчийн интерфэйс хөгжүүлсэн. TypeScript хэл ашигласан нь хэрэглэгчийн кодын аюулгүй байдлыг хангаж, Tailwind CSS болон Shadcn/UI компонентууд ашигласан нь хурдан, уян хатан дизайн хийх боломжийг олгосон. TradingView виджетүүд ашиглан МХБ-аас гадна олон улсын хувьцаануудын график харуулна.

4.2.2 API Gateway давхарга

Express.js 4.18 ашиглан RESTful API бүтээсэн. API Gateway нь хэрэглэгчийн бүх хүсэлтийг хүлээн авч, Kafka руу үйл явдал (kafka event) дамжуулах үүрэгтэй. Мөн JWT токен ашиглан хэрэглэгчийн танин баталгаажуулалт хийж, Server-Sent Events (SSE) ашиглан бодит цагийн хариултыг хэрэглэгч рүү дамжуулна.

Гол endpoints:

- `POST /api/agent/query` - Хэрэглэгчийн асуултыг хүлээн авч, төлөвлөгч агент руу илгээх
- `GET /api/agent/response/:id` - Агентын хариултыг polling хэлбэрээр авах
- `GET /api/agent/stream/:id` - SSE ашиглан бодит цагийн хариулт авах
- `POST /api/agent/analyze-watchlist` - Хэрэглэгч өөрийн сонирхсон МХБ-ийн хувьцаануудын шинжилгээг авах

4.2.3 Apache Kafka давхарга

Apache Kafka 3.5 нь системийн төв мэдээллийн сүлжээ болж, бүх агентууд үүгээр дамжин харилцана. Zookeeper 3.8 нь Kafka кластерийн байршуулалтыг удирдана.

4.2.4 Агентуудын давхарга

Таван агент бие даасан микросервис хэлбэрээр Docker контейнерт ажиллана. Агент бүр өөрийн Kafka consumer, producer-тэй бөгөөд бусад агентуудаас хараат бус ажиллах боломжтой.

4.2.5 Өгөгдлийн давхарга

PostgreSQL 16 нь хэрэглэгчид, хувьцааны мэдээлэл, арилжааны түүх, агентуудын хариултыг хадгална. Redis 7 нь session болон response caching-д ашиглагдана.

4.3 Агентуудын дэлгэрэнгүй тайлбар

Энэхүү хэсэгт агент бүрийн үүрэг, ажиллах зарчим, Kafka сэдвүүдтэй харилцах байдлыг дэлгэрэнгүй тайлбарлана.

4.3.1 Төлөвлөгч агент (Orchestrator Agent)

Төлөвлөгч агент нь системийн төв тархи болж, хэрэглэгчийн хүсэлтийг ангилж, зохих агент руу чиглүүлэх үүрэгтэй. Энэ нь ReAct (Reasoning and Acting) загварын гол бүрэлдэхүүн юм.

Гол үүргүүд:

1. Intent Classification: Google Gemini 2.5 Flash ашиглан хэрэглэгчийн асуултыг portfolio, market_analysis, news, risk_assessment, historical_analysis, general_query гэсэн ангилалд хуваана

2. Complexity Detection: Асуулт энгийн эсвэл олон агент шаардсан нарийн төвөгтэй эсэхийг тодорхойлно

3. User Profile Fetching: PostgreSQL-ээс хэрэглэгчийн investment_goal, risk_tolerance, preferred_industries мэдээллийг авч, хувийн зөвлөмж өгөхөд ашиглана

4. RAG Context Request: Шаардлагатай бол мэдлэгийн агент руу хүсэлт илгээж, нэмэлт контекст авна

5. Routing: Энгийн асуултыг шууд agent.tasks руу, нарийн төвөгтэй асуултыг planning.tasks руу илгээнэ

Процессийн урсгал (Код 4.1):

Төлөвлөгч агент нь user.requests topic-оос хүсэлт хүлээн авч, Gemini AI ашиглан intent ангилна. Дараа нь хэрэглэгчийн профайл авч, нарийн төвөгтэй байдлыг шалгаад, тохирох агент руу чиглүүлнэ.

```
const intent = await intentClassifier.classify(query);
const userProfile = await getUserProfile(userId);
const complexity = await complexityDetector.detect(query);

if (complexity.level === 'simple') {
  await routeToAgent(requestId, userId, intent, query);
```

```
} else {
  await routeToPlanner(requestId, userId, intent, query);
}
```

Код 4.1: Orchestrator Agent main flow

Kafka topics:

- Subscribe: user.requests, knowledge.results
- Publish: agent.tasks, planning.tasks, knowledge.queries, monitoring.events

4.3.2 Хөрөнгө оруулалтын агент (Investment Agent)

Хөрөнгө оруулалтын агент нь МХБ хувьцааны дүн шинжилгээ хийж, хэрэглэгчийн профайлд суурилсан монгол хэл дээрх хувийн зөвлөмж өгөх үүрэгтэй.

Гол үүргүүд:

1. МХБ өгөгдөл татах: PostgreSQL-ээс mse_trading_status (сүүлийн үнэ) болон mse_trading_history (түүхэн өгөгдөл) хүснэгтээс мэдээлэл авна
2. Хувийн зөвлөмж үүсгэх: Хэрэглэгчийн investmentGoal, riskTolerance, preferredIndustries мэдээлэлд суурилан AI промпт бэлдэнэ
3. Монгол хариулт үүсгэх: Gemini 2.5 Flash ашиглан монгол хэл дээр товч, мэргэжлийн хариулт үүсгэнэ
4. Хариулт хадгалах: agent_responses_cache хүснэгтэд хариултыг хадгалж, дараа нь polling хийхэд ашиглана

Дэмжигдэх үйлдлүүд:

- `analyze_portfolio` - Багцын шинжилгээ
- `provide_advice` - Хөрөнгө оруулалтын зөвлөмж
- `analyze_market` - Зах зээлийн шинжилгээ

- `analyze_watchlist` - Өөрийн сонирхсон хувьцаануудын шинжилгээ

Хувийн зөвлөмжийн логик (Код 4.2):

Хэрэглэгчийн профайлаас контекст бүрдүүлж, Gemini AI руу промпт илгээнэ. Эрсдлийн хүлээцтэй байдал "Low" бол аюулгүй хувьцаа, "High" бол өндөр өсөлттэй хувьцаа санал болгоно.

```
const personalizationContext = `
User Profile:
- Goal: ${userProfile.investmentGoal}
- Risk: ${userProfile.riskTolerance}
- Industries: ${userProfile.preferredIndustries}
`;

const prompt = `MSE Analyst. MONGOLIAN, BRIEF.
${personalizationContext}
MSE Data: ${mseData}
Provide advice.`;

const result = await model.generateContent(prompt);
```

Код 4.2: Personalization context building

4.3.3 Мэдээний агент (News Agent)

Мэдээний агент нь Finnhub API-аас олон улсын санхүүгийн мэдээ татаж, утга зүйн шинжилгээ хийж, хураангуй бэлдэх үүрэгтэй.

Гол үүргүүд:

1. **Мэдээ татах:** Finnhub API-аас сүүлийн 7 хоногийн мэдээг татна
2. **Sentiment analysis:** Gemini AI ашиглан мэдээ бүрийг positive, negative, neutral гэж ангилна

3. **Хураангуй үүсгэх:** Олон мэдээг нэгтгэж, зах зээлийн ерөнхий мэдрэмжийн хураангуй бэлдэнэ

4.3.4 Мэдлэгийн агент (Knowledge Agent)

Мэдлэгийн агент нь RAG (Retrieval-Augmented Generation) системийн үүргийг гүйцэтгэж, мэдлэгийн сангаас холбогдолтой мэдээллийг хайж олно.

Гол үүргүүд:

1. Өгөгдлийн сан: PostgreSQL-ийн knowledge_base хүснэгтээс компанийн профайл, бизнес дүрэм, агентуудын чадварын мэдээллийг ачаална
2. Утга зүйн хайлт: Түлхүүр үгийн тохирол болон metadata фильтер ашиглан хамгийн холбогдолтой баримтуудыг олно
3. Контекст дамжуулах: Олдсон баримтуудыг knowledge.results сэдвээр буцаана

Өгөгдлийн сангийн мэдээллүүд:

- МХБ компаниудын профайл (APU, TDB гэх мэт)
- Арилжааны дүрэм (арилжааны цаг, хязгаарлалт)
- Эрсдлийн менежментийн зөвлөмж
- Агент бүрийн чадварын тайлбар (төлөвлөгч агентд туслах)

4.3.5 PyFlink төлөвлөгч агент (Flink Planner)

PyFlink төлөвлөгч нь нарийн төвөгтэй, олон алхамт даалгаврыг төлөвлөж, гүйцэтгэлийн дараалал үүсгэх үүрэгтэй.

Гол үүргүүд:

1. Гүйцэтгэлийн төлөвлөгөө үүсгэх: Gemini AI эсвэл дүрмэнд суурилсан логик ашиглан олон алхамт төлөвлөгөө боловсруулна

2. Агентуудад даалгавар хуваарилах: Төлөвлөгөөний алхам бүрийг тохирох агент руу agent.tasks сэдвээр илгээнэ

3. Параллель гүйцэтгэл: Хамааралгүй алхмуудыг зэрэг гүйцэтгэх боломжийг олгоно

4.4 Kafka сэдвүүд ба өгөгдлийн урсгал

Систем нь 8 гол Kafka сэдэв ашиглана. Хүснэгт 4.1-д сэдэв бүрийн үүргийг тайлбарлав.

Хүснэгт 4.1: Kafka сэдвүүдийн тайлбар

Сэдэв	**Үүрэг**
user.requests	Хэрэглэгчийн асуултууд API Gateway-ээс төлөвлөгч агент руу
agent.tasks	Зохион байгуулагчаас мэргэшсэн агентууд руу даалгавар
agent.responses	Агентуудын хариулт буцаах
knowledge.queries	Мэдлэгийн агент руу RAG хайлтын хүсэлт
knowledge.results	Мэдлэгийн агентын хайлтын үр дүн
planning.tasks	PyFlink Planner руу нарийн төвөгтэй даалгавар
execution.plans	Төлөвлөгчийн үүсгэсэн гүйцэтгэлийн төлөвлөгөө
monitoring.events	Системийн мониторинг, лог бичлэг

Өгөгдлийн урсгалын жишээ:

Хэрэглэгч “APU хувьцааны шинжилгээ хийж өгнө үү” гэж асуухад дараах урсгал өрнөнө:

1. Frontend → API Gateway: HTTP POST /api/agent/query

2. API Gateway → Kafka: user.requests сэдэв рүү мессеж бичих

3. Orchestrator: user.requests-аас уншиж, intent ангилах (portfolio)

4. Orchestrator: Хэрэглэгчийн профайл PostgreSQL-ээс авах

5. Orchestrator → Kafka: agent.tasks сэдэв рүү даалгавар илгээх
6. Investment Agent: agent.tasks-аас уншиж, МХБ өгөгдөл татах
7. Investment Agent: Gemini AI-аар монгол хариулт үүсгэх
8. Investment Agent → Kafka: agent.responses сэдэв рүү хариулт бичих
9. Investment Agent → PostgreSQL: agent_responses_cache-д хадгалах
10. API Gateway: Polling эсвэл SSE-ээр хариултыг Frontend руу дамжуулах

4.5 Өгөгдлийн сангийн бүтэц

PostgreSQL 16 өгөгдлийн санд дараах гол хүснэгтүүд байна:

Хүснэгт 4.2: Өгөгдлийн сангийн хүснэгтүүд

Хүснэгт	**Агуулга**
users	Хэрэглэгчийн мэдээлэл, профайл (investment_goal, risk_tolerance, preferred_industries)
watchlists	Хэрэглэгчийн ажиглаж болох хувьцааны багцууд
watchlist_items	Жагсаалт доторх хувьцаанууд (is_mse флагтай)
mse_companies	МХБ-д бүртгэлтэй компаниуд (symbol, name, sector)
mse_trading_status	Сүүлийн арилжааны үнэ, хэмжээ
mse_trading_history	2015-2024 оны арилжааны түүх
knowledge_base	RAG системийн мэдлэгийн сан
agent_responses_cache	Агентуудын хариултын кэш
monitoring_events	Системийн мониторингийн лог

4.6 Хэрэглэгчийн шаардлага

Систем хэрэглэгчдэд дараах боломжуудыг олгоно:

4.6.1 Бүртгэл ба нэвтрэлт

Хэрэглэгч бүртгүүлэхдээ хөрөнгө оруулалтын профайл бөглөнө:

- Хөрөнгө оруулалтын зорилго (Growth, Income, Balanced, Conservative)
- Эрсдлийн хүлээцтэй байдал (Low, Medium, High)
- Сонирхсон салбарууд (Technology, Finance, Mining гэх мэт)

Бүртгэлийн дараа хиймэл оюун ашиглан хувийн мэндчилгээний э-мэйл монгол хэлээр илгээгдэнэ.

4.6.2 Сонирхсон хувьцааны жагсаалт

Хэрэглэгч дэлхийн болон МХБ хувьцааг ажиглах жагсаалтад нэмж болно. МХБ хувьцааны хувьд хиймэл оюуны шинжилгээ авах боломжтой.

4.6.3 AI чатбот

Хэрэглэгч монгол эсвэл англи хэлээр асуулт асууж болно. Жишээ асуултууд:

- “APU хувьцааны сүүлийн үнэ ямар байна вэ?”
- “Миний ажиглаж буй хувьцаануудыг шинжилж өгнө үү”
- “Зах зээлийн ерөнхий байдал ямар байна?”

4.6.4 Өдөр тутмын мэдээ

Хэрэглэгч өдөр бүр өөрийн ажиглаж буй хувьцааны мэдээг э-мэйлээр авах боломжтой. Finnhub API-аас мэдээ татаж, Gemini AI-аар хураангуйлж бүх хэрэглэгч рүү илгээнэ. Хэрэв сонирхсон хувьцаа байхгүй бол сүүлийн үеийн дурын мэдээний хураангуйг илгээнэ.

4.7 Технологийн стек

4.7.1 Frontend технологи

Хүснэгт 4.3: Frontend технологийн стек

Технологи	**Хэрэглээ**
Next.js 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
TradingView	Олон улсын хувьцааны график, виджетүүд

4.7.2 Backend технологи

Хүснэгт 4.4: 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-ийн message broker
Zookeeper 3.8	Kafka кластерийн зохицуулалт
PostgreSQL 16	Үндсэн өгөгдлийн сан
Redis 7	Session болон response caching
Docker	Контейнержуулалт, бие даасан байршуулалт

4.7.3 Хиймэл оюуны технологи

Хүснэгт 4.5: Хиймэл оюуны технологи

Технологи	**Хэрэглээ**
Google Gemini 2.5 Flash	Intent classification, хариулт үүсгэх, sentiment analysis
Finnhub API	Олон улсын санхүүгийн мэдээний эх сурвалж
Nodemailer	Э-мэйл илгээх сервис

4.8 Туршилтын үр дүн

4.8.1 Гүйцэтгэлийн хэмжилт

Системийн гүйцэтгэлийг хэмжихэд дараах үр дүн гарсан, мөн бүх процессууд докерт нийт 123mb санах ойн хэмжээтэй байна:

Хүснэгт 4.6: Гүйцэтгэлийн хэмжилт

Үйлдэл	**Хугацаа**
PostgreSQL-ээс МХБ өгөгдөл татах	50-100ms
Kafka мессеж дамжуулалт	5-10ms
API Gateway endpoint хариу	200-500ms
Gemini AI хариулт үүсгэх	3-8 секунд
Бүрэн AI шинжилгээний процесс	5-12 секунд

4.8.2 Амжилттай туршигдсан функцууд

Хэрэгжүүлэлтийн үр дүнг тайлангийн хавсаргалтад оруулав. Дараах функцууд бүрэн ажиллаж байна:

- Хэрэглэгчийн бүртгэл, нэвтрэлт, JWT таниулалт
- Сонирхсон хувьцааны жагсаалт үүсгэх, хувьцаа нэмэх/хасах
- МХБ хувьцааны AI шинжилгээ (монгол хэлээр)
- Хувийн зөвлөмж (хэрэглэгчийн профайлд суурилсан)

- Бүртгэлийн AI боловсруулсан мэдэгдэл авах э-мэйл (монгол хэлээр)
- Өдөр тутмын мэдээний AI боловсруулсан э-мэйл
- Агентуудын мониторинг API
- SSE болон polling-аар бодит цагийн хариулт авах

4.8.3 Байршуулалт

Docker болон Docker Compose ашиглан бүх сервисийг контейнержуулсан. Агент бүр өөрийн Docker контейнерт ажиллаж, бие даан өргөжих боломжтой. Frontend-ийг Vercel Hobby Plan ашиглан байршуулсан.

Хаяг: `https://stock-tracker-app-topaz.vercel.app/`

4.9 Бүлгийн дүгнэлт

Энэхүү бүлэгт хиймэл оюун агентуудыг микросервис архитектурт нэвтрүүлсэн демо системийн хэрэгжүүлэлтийг дэлгэрэнгүй тайлбарлав. Төлөвлөгч агент нь ReAct загварын гол бүрэлдэхүүн болж, хэрэглэгчийн хүсэлтийг ангилж, хэрэглэгчийн профайл авч, зохих агент руу чиглүүлэх үүргийг амжилттай гүйцэтгэж байна.

Хөрөнгө оруулалтын агент нь МХБ өгөгдөлд суурилан хувийн зөвлөмж монгол хэлээр өгч байгаа нь системийн гол онцлог юм. Мэдээний агент нь Finnhub API-аас мэдээ татаж, утга зүйн шинжилгээ хийж байна. Мэдлэгийн агент нь RAG системийн үүргийг гүйцэтгэж, нэмэлт контекст өгч байна.

Apache Kafka ашигласан нь агентуудыг тархмал, бие даасан байлгаж, хэвтээ өргөжих боломжийг олгосон. Ийнхүү онолын хэсэгт судалсан микросервис архитектур, EDA, хиймэл оюун агентуудын зарчмуудыг практикт амжилттай хэрэгжүүлсэн.

Дүгнэлт

Энэхүү судалгааны ажлаар хиймэл оюун агентуудыг микросервис архитектурт нэвтрүүлэх боломжийг онол, хэрэгжүүлэлтийн хувьд судалж, үзэгдэлд суурилсан архитектурт зохиомжийг санал болголоо. Судалгааны хүрээнд хиймэл оюуны инженерчлэл, суурь загвар, агентуудын онолыг гүнзгийрүүлэн судалж, ижил төстэй системүүдтэй харьцуулан санал болгож буй зохиомжийн шинэлэг байр суурийг тодорхойлсон.

Онолын судалгааны явцад хэл загвараас суурь загвар хүртэлх хөгжлийн түүх, RAG систем, агентуудын төлөвлөлт (ReAct) зэрэг ойлголтуудыг авч үзсэн. Мөн микросервис архитектурын давуу тал болох технологийн уян хатан байдал, хэвтээ өргөжих боломж нь хиймэл оюун агентуудыг тархмал сервис болгон хөгжүүлэхэд тохиромжтой болохыг тодорхойлсон бол монолит агентуудын NxM нягт холболтын асуудлыг шийдэхийн тулд үзэгдэлд суурилсан архитектур шаардлагатай болохыг бататгав.

Судалгаагаа бататгахын тулд Монголын Хөрөнгийн Биржийн бодит өгөгдөлд тулгуурлан олон агент бүхий демо системийг хөгжүүлсэн. Уг системд төлөвлөгч, хөрөнгө оруулалтын, мэдээний, мэдлэгийн, PyFlink төлөвлөгч гэсэн таван агентыг бие даасан микросервис хэлбэрээр хэрэгжүүлж, Apache Kafka ашиглан агентуудын асинхрон харилцааг зохион байгуулсан. Хөрөнгө оруулалтын агент нь хэрэглэгчийн профайлд суурилан монгол хэлээр хөрөнгө оруулалтын хувийн зөвлөмж өгөх боломжийг бүрдүүлсэн бол мэдлэгийн агент нь RAG системийн үүргийг гүйцэтгэж, суурь загварын дотоод мэдээллээс гадна нэмэлт мэдээлэл нэмэх боломжийг бүрдүүлж байна.

Сэдвийн цар хүрээ өргөн, технологийн хувьд нарийн төвөгтэй хэдий ч онолын судалгааг бүтэн хэрэгжүүлэлттэй амжилттай хослуулж чадсан. Суурь загварын хоцрогдол, өртөг зэрэг хязгаарлалтууд байгаа бөгөөд цаашид агентуудын санах ойн удирдлага, олон агентын хамтын ажиллагаа, монгол хэлний боловсруулалтын оновчлол зэрэг чиглэлээр судалгааг үргэлжлүүлэх боломжтой.

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] What is ReAct Agent? https://www.ibm.com/think/topics/react-agent

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

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

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

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

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

[11] Gemini. "Gemini API Reference". https://ai.google.dev/gemini-api/docs

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

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

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

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

[16] Apache Kafka Documentation. "Apache Kafka: A Distributed Streaming Platform". https://kafka.apache.org/documentation/

[17] Apache Flink Documentation. "Stateful Computations over Data Streams". https://flink.apache.org/

[18] Temporal Documentation. https://docs.temporal.io/

[19] Inngest Documentation. https://www.inngest.com/

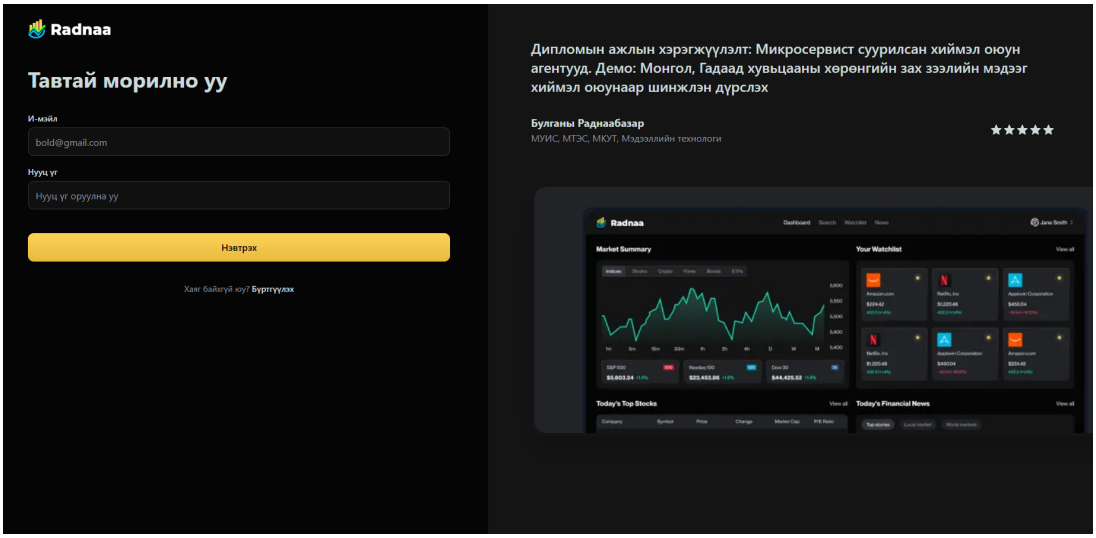
[20] Wolff, Eberhard. *Microservices: Flexible Software Architecture*. Addison-Wesley Professional, 2016.

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

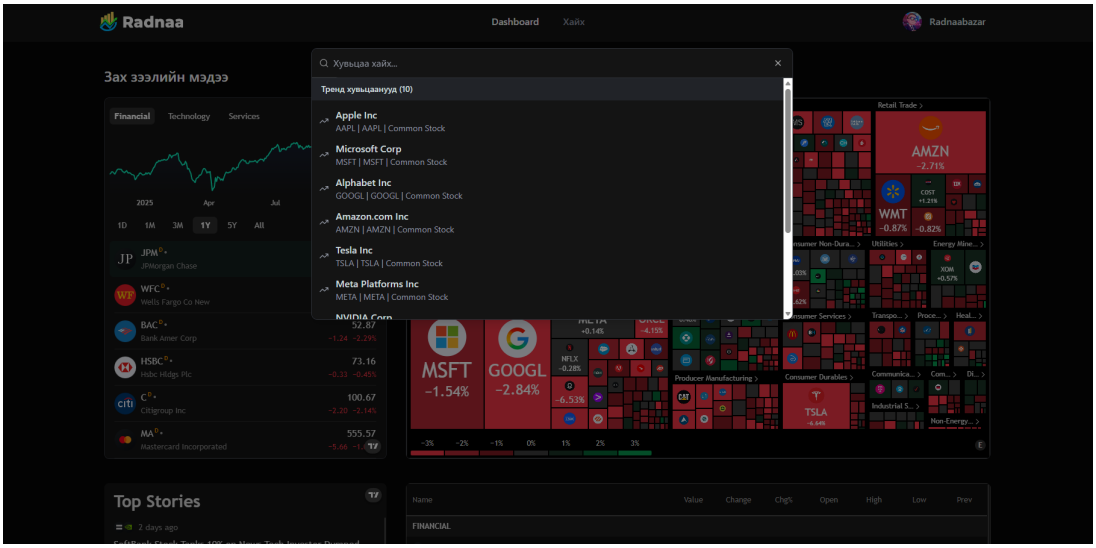
[22] Anthropic. "Model Context Protocol: A Universal Standard for AI Integration", 2024. https://www.anthropic.com/news/model-context-protocol

[23] AWS. "What are Microservices?". https://aws.amazon.com/microservices/

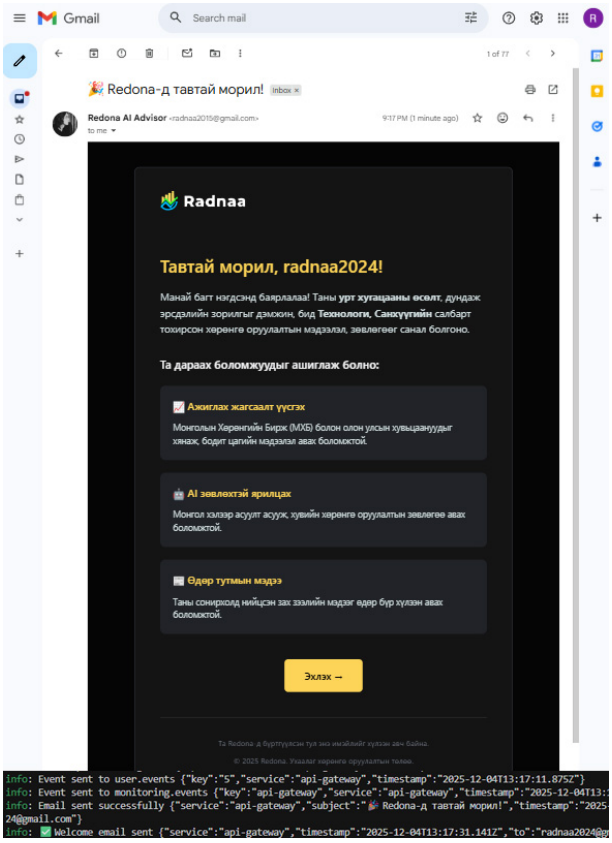
А. ХЭРЭГЖҮҮЛЭЛТИЙН ҮР ДҮН



Зураг А.1: Нэвтрэх хуудас

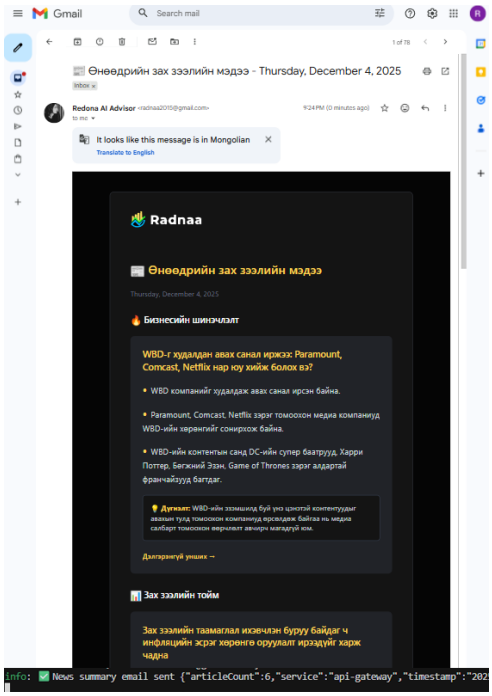


Зураг А.2: Нүүр хуудас



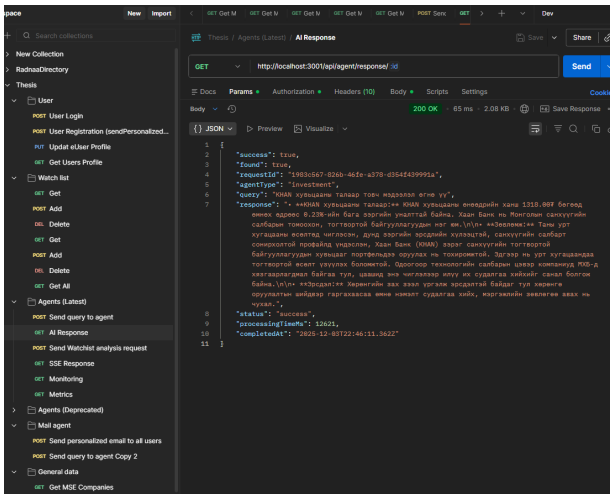
Зураг А.3: AI Боловсруулсан бүртгэлийн мэдэгдлийг э-мэйлээр авах

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 илгээнэ. Зургийн ёроолд API Лог үр дүнг доор үзүүлэв.



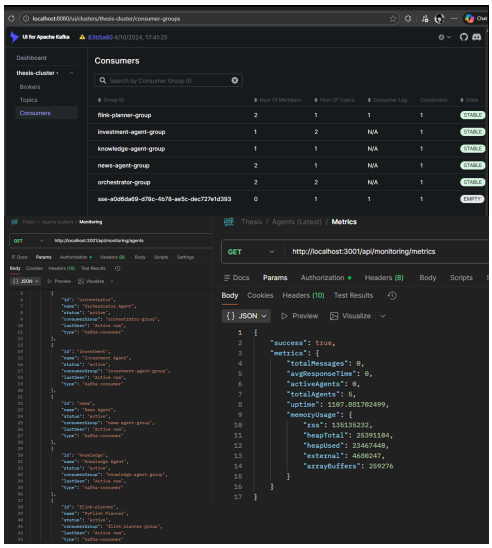
Зураг А.4: AI боловсруулсан өдөр тутмын мэдээний э-мэйл авах

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.



Зураг А.5: Хөрөнгө оруулалтын агентийн хэрэглэгчийн хувийн зөвлөмж авах

Төлөвлөгч агент нь ”KHAN хувьцааны талаар товч мэдээлэл өгнө үү” гэсэн хүсэлтээр хэрэглэгчийн мэдээлэл болон хаан банкны хувьцааны мэдээллийг нэгтгэж Gemini AI-аар шинжилгээ хийнэ. Хэрэглэгчийн мэдээлэлд эрсдлийн хүлээцтэй байдал, зорилго зэрэг мэдээ байх ба агент нь үүний үндсэнд тохирсон зөвлөгөө өгнө.



Зураг А.6: Агентийн ажиллаж байгаа процессуудын мониторинг

Байршуулалтаа docker дээр хийсэн ба, бүх агент, сервис ажиллаж байгаа эсэхийг мониторингийг дээрх зургаар дүрсэлж байна. Kafka UI дээр бүх topic-үүд, кластер, хүлээн авагч нь Stable буюу асуудалгүй ажиллаж байна. Postman-аар агент хоорондын харилцааг мөн системийн нөөцийн ашиглалтыг шалгасан. Нийт процессүүд ойролцоогоор idle үед 123mb ашиглаж байна.

В. КОДЫН ЖИШЭЭ

Энэхүү хавсралтад системийн гол бүрэлдэхүүн хэсгүүдийн кодын жишээг оруулав.

В.1 Төлөвлөгч агент: Хэрэглэгчийн хүсэлт боловсруулах

Төлөвлөгч агент нь user.requests topic-оос хүсэлт хүлээн авч, intent ангилж, хэрэглэгчийн профайл авч, зохих агент руу чиглүүлнэ.

```
private async processUserRequest(payload: any): Promise<void> {
  const { requestId, userId, query, type, context } = payload;

  // Step 1: Classify intent using Gemini AI
  const intent = type || await intentClassifier.classify(query);
  logger.info(`Intent classified: ${intent}`, { requestId });

  // Step 2: Fetch user profile for personalization
  let userProfile = null;
  if (userId && agentType === 'investment') {
    userProfile = await this.getUserProfile(userId);
  }

  // Step 3: Detect complexity
  const complexity = await complexityDetector.detect(query);

  // Step 4: Route based on complexity
  if (complexity.level === 'simple') {
    await this.routeToAgent(requestId, userId, intent, query);
  } else {
    await this.routeToPlanner(requestId, userId, intent, query);
  }
}

private async getUserProfile(userId: string): Promise<any> {
  const result = await db.query(
    `SELECT investment_goal, risk_tolerance,
           preferred_industries
     FROM users WHERE id = $1`,
    [userId]
  );
  return result.rows[0] || {};
}
```

Код В.1: Orchestrator Agent - Request Processing

В.2 Хөрөнгө оруулалтын агент: Хувийн зөвлөмж үүсгэх

Хэрэглэгчийн профайлд суурилан МХБ хувьцааны шинжилгээ хийж, монгол хэлээр зөвлөмж өгнө.

```
async function generateAIResponse(action: string, payload: any) {
  const { userId, query, userProfile } = payload;
  const symbols = payload.context?.symbols || [];

  // Fetch MSE data from PostgreSQL
```

```
  const mseData = symbols.length > 0
    ? await getMSEData(symbols)
    : await getMSEData();

  // Build personalization context
  let context = '';
  if (userProfile) {
    context = `User Profile:
- Goal: ${userProfile.investmentGoal}
- Risk Tolerance: ${userProfile.riskTolerance}
- Preferred Industries: ${userProfile.preferredIndustries}

IMPORTANT: If risk is "Low", recommend safe stocks.
If risk is "High", recommend growth stocks.`;
  }

  // Generate response with Gemini AI
  const prompt = `MSE Analyst. MONGOLIAN, BRIEF (150 words).
${context}
MSE Data: ${JSON.stringify(mseData.slice(0, 8))}
Query: ${query}
Provide personalized investment advice.`;

  const result = await model.generateContent(prompt);
  return result.response.text();
}
```

Код B.2: Investment Agent - Personalized Response

B.3 API Gateway: Kafka Event Publishing

Хэрэглэгчийн асуултыг Kafka-ийн user.requests topic руу илгээнэ.

```
router.post('/query', async (req: Request, res: Response) => {
  const { query, type, context } = req.body;
  const userId = getUserId(req);
  const requestId = uuidv4();

  // Send to Kafka user.requests topic
  await kafkaService.sendEvent('user.requests', requestId, {
    requestId,
    userId,
    timestamp: new Date().toISOString(),
    query,
    type: type || undefined,
    context: context || {},
  });

  res.json({
    success: true,
    requestId,
    message: 'Query submitted successfully',
    pollEndpoint: `/api/agent/response/${requestId}`,
  });
});
```

Код B.3: API Gateway - Kafka Event Publishing

B.4 PyFlink Planner: Execution Plan Generation

Нарийн төвөгтэй даалгаврыг олон алхамт төлөвлөгөө болгон задална.

```python
def generate_execution_plan_with_gemini(task: dict) -> dict:
    query = task.get('query', '')
    intent = task.get('intent', '')

    prompt = f"""Generate execution plan for financial AI.
Query: {query}
Intent: {intent}

Available Agents:
1. knowledge - RAG for MSE company info
2. investment - Portfolio analysis, recommendations
3. news - Financial news and sentiment

Generate JSON with steps array. Max 3 steps."""

    response = model.generate_content(prompt)
    plan = json.loads(response.text)

    # Send tasks to each agent
    for step in plan.get('steps', []):
        agent_task = {
            "taskId": str(uuid.uuid4()),
            "agentType": step.get('agent'),
            "action": step.get('action'),
            "payload": step.get('params', {}),
        }
        producer.send('agent.tasks',
                      value=json.dumps(agent_task))

    return plan
```

Код B.4: PyFlink Planner - Plan Generation

B.5 Мэдээний агент: Sentiment Analysis

Finnhub API-аас мэдээ татаж, Gemini AI-аар sentiment шинжилгээ хийнэ.

```javascript
async function analyzeSentiment(headline: string, summary: string) {
  const prompt = `Analyze sentiment of this financial news:

Headline: ${headline}
Summary: ${summary}

Respond with ONLY ONE WORD: positive, negative, or neutral.`;

  const result = await model.generateContent(prompt);
  const sentiment = result.response.text().trim().toLowerCase();

```

```
  if (['positive', 'negative', 'neutral'].includes(sentiment)) {
    return sentiment;
  }
  return 'neutral';
}

async function handleNewsTask(message: any) {
  const news = await fetchNews(message.payload.symbol);

  // Analyze sentiment for each article
  const processedNews = [];
  for (const item of news.slice(0, 5)) {
    const sentiment = await analyzeSentiment(
      item.headline, item.summary
    );
    processedNews.push({ ...item, sentiment });
  }

  // Generate summary
  const summary = await model.generateContent(
    `Summarize these news: ${JSON.stringify(processedNews)}`
  );

  return { news: processedNews, summary: summary.text() };
}
```

Код В.5: News Agent - Sentiment Analysis