

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 10 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 10 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

УДИРТГАЛ	1
Нэр томъёоны тайлбар	3
1. ОНОЛЫН ХЭСЭГ	10
1.1 Хиймэл оюуны инженерчлэл гэж юу вэ	10
1.2 Суурь моделийн хөгжил	11
1.3 Хиймэл оюуны агент ба бизнесийн үйл ажиллагаа	12
1.4 Суурь моделийн сургалт	13
1.5 Зааврын инженерчлэл	15
1.6 Хайлтаар нэмэгдүүлсэн үүсгэлт (RAG)	16
1.7 Хиймэл оюуны агентууд	19
1.8 Жишээ хиймэл оюунт программ хангамжийн зохиомж	24
1.9 Бүлгийн дүгнэлт	25
2. МИКРОСЕРВИС АРХИТЕКТУР	27
2.1 Монолитаас микросервис рүү	27
2.2 Микросервисийн давуу тал	28
2.3 Микросервисийн сорилтууд	29
2.4 Микросервис хоорондын харилцаа	29
2.5 Үйл явдлаар удирдагдах архитектур буюу EDA	32
2.6 Бүлгийн дүгнэлт	36
3. АСУУДЛЫН ТОДОРХОЙЛОЛТ БА ШИЙДЭЛ	39
3.1 Монолит агентуудын эрсдэл	39
3.2 Микросервис агентуудын шийдэл	40
3.3 Ижил төстэй системүүдийн судалгаа	44
3.4 Бүлгийн дүгнэлт	46

4. ХЭРЭГЖҮҮЛЭЛТ	47
4.1 Удиртгал	47
4.2 Төслийн зорилго	47
4.3 Системийн функционал	48
4.4 Хэрэглэгчийн шаардлага	50
4.5 Системийн архитектур	51
4.6 Технологийн стек	54
4.7 Хэрэглээний тохиолдол	55
4.8 Хөгжүүлэлтийн явц	58
4.9 Бүлгийн дүгнэлт	59
ДҮГНЭЛТ	61
НОМ ЗҮЙ	63
ХАВСРАЛТ	65
А. ӨӨРИЙН МЭДЭЭЛЭЛД ТОХИРСОН БАЙДЛААР БҮРТГҮҮЛЭЛТИЙН Э-МЭЙЛ АВАХ	66
В. ӨӨРИЙН МЭДЭЭЛЭЛД ТОХИРСОН БАЙДЛААР ӨДӨР ТУТМЫН Э-МЭЙЛ АВАХ	67

ЗУРГИЙН ЖАГСААЛТ

1.1	Хиймэл оюуны инженерчлэл ба машин сургалтын инженерчлэл	10
1.2	RAG-ийн бүтэц	17
1.3	Агентын бүрэлдэхүүн хэсгүүд	20
1.4	Агентын төлөвлөлт	21
1.5	Хиймэл оюунт аппликейшн	24
2.1	Синхрон микросервис	30
2.2	Синхрон микросервисийн NxM холбоо	31
2.3	Асинхрон микросервис	31
2.4	Үйл явдлаар удирдагдах архитектурт микросервис	32
3.1	Монолит агентын архитектур: NxM нягт холбоос	40
3.2	Үйл явдлаар удирдагдах олон агентын систем	41
3.3	Бодит хэрэгжүүлсэн архитектур: Kafka-Flink суурилсан олон агентын систем	43
4.1	Хэрэгжүүлсэн системийн архитектур	52
A.1	RAG-ийн бүтэц	66
B.1	RAG-ийн бүтэц	67

ХҮСНЭГТИЙН ЖАГСААЛТ

1	Англи хэлнээс орчуулсан нэр томъёо	3
2	Товчилсон үгс	5
3	Техникийн нэр томъёо	7
3.1	Ижил төстэй системүүдийн харьцуулалт	44
3.2	Бидний санал болгож буй загварын онцлог	45
4.1	Хэрэгжүүлсэн агентуудын үүрэг	53
4.2	Frontend технологийн стек	54
4.3	Backend технологийн стек	54
4.4	Хиймэл оюуны технологи	55

Кодын жагсаалт

УДИРТГАЛ

Сүүлийн жилүүдэд хиймэл оюуны салбар дахь технологийн хурдацтай хөгжил нь програм хангамж хөгжүүлэлтийн арга барилд үндсэндээ өөрчлөлт авчирсан. Тухайлбал, суурь модел 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 модел 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. Өмнө нь машин сургалт загвар бэлтгэхэд өндөр мэргэжлийн ур чадвар болон асар их өгөгдөл шаардлагатай байсан бол одоо бэлэн моделийг ашиглан аппликейшн хөгжүүлж 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 моделийн 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 моделийн онол, агентуудын төлөвлөлт, RAG болон эдгээрийг микросервис архитектурт хэрхэн нэгтгэх талаар авч үзэх юм.

Нэр томъёоны тайлбар

Англи хэлнээс орчуулсан нэр томъёо

Хүснэгт 1: Англи хэлнээс орчуулсан нэр томъёо

Англи нэр	Монгол орчуулга
AI Engineering	Хиймэл оюуны инженерчлэл
Foundation Model	Суурь модел
Language Model	Хэлний модел
Masked Language Model	Далдлагдсан хэлний модел
Large Language Model	Том хэлний модел
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

Хүснэгт 1: Англи хэлнээс орчуулсан нэр томъёо (үргэлжлэл)

Англи нэр	Монгол орчуулга
Token	Токен (хэлний моделд тэмдэгт мөрийн таслалын нэгж)
Inference	Гаргалгаа (моделоос 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 (моделийн бүтээлч чанарын параметр)
Hallucination	Төөрөгдөл (моделийн 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

Хүснэгт 1: Англи хэлнээс орчуулсан нэр томъёо (үргэлжлэл)

Англи нэр	Монгол орчуулга
Coupling	Хамаарал
Decoupling	Салангид байдал
Synchronous communication	Синхрон холбоо
Asynchronous communication	Асинхрон холбоо
Partition	Хэсэглэл
Topic	Сэдэв

Товчилсон үгс

Хүснэгт 2: Товчилсон үгс

Товчлол	Нэр томъёо	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 Encoder Representations from Transformers	Transformer архитектурт суурилсан хэлний модел
GPT	Generative Pre-trained Transformer	Урьдчилан сурсан үүсгэгч Transformer
NLP	Natural Language Processing	Байгалийн хэлний боловсруулалт
ML	Machine Learning	Машин сургалт

Хүснэгт 2: Товчилсон үгс (үргэлжлэл)

Товчлол	Нэр томъёо	Монгол тайлбар
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 томъёоны 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, MXБ	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

Хүснэгт 3: Техникийн нэр томъёо

Нэр томъёо	Тайлбар
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 архитектурт нэвтрүүлэх боломжийг онол, практикийн хувьд судалж, үйл явдлаар удирдагдах архитектур ашиглан уян хатан, өргөжих боломжтой систем бүтээх зохиомж санал болгох юм. Санал болгож буй зохиомжийн үр ашигтай байдлыг баталгаажуулахын тулд монгол, гадаад хөрөнгийн зах зээлийн өгөгдөлд суурилсан практик демо систем хөгжүүлж, туршиж үзнэ. Энэ нь хиймэл оюуны технологи болон орчин үеийн програм хангамжийн архитектурын уялдааг судалж, практикт хэрэглэх боломжийг харуулна.

Зорилт:

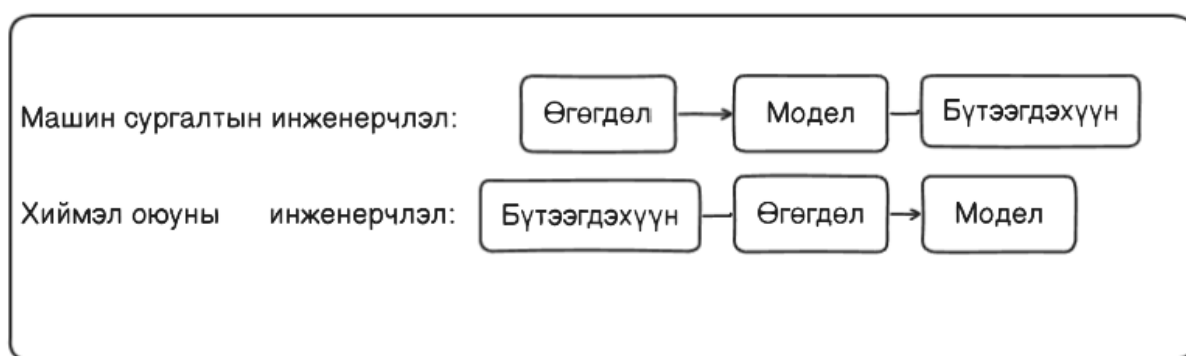
1. Хиймэл оюуны инженерчлэлийн үндсэн онол, суурь моделийн хөгжил, зааврын инженерчлэл, RAG системийн онолыг тайлбарлах. Цаашлаад яагаад олон агент систем нь бизнесд зохимжтой вэ гэдгийг тайлбарлана.
2. Хиймэл оюун агентуудын архитектур, бүрэлдэхүүн хэсгүүд, төлөвлөлтийн механизм, хэрэглүүрүүдийн ашиглалтыг судлах
3. Микросервис архитектурын давуу болон сул талуудыг тодорхойлж, үйл явдлаар удирдагдах архитектурын давуу тал болон шаардлагыг тайлбарлах
4. Хиймэл оюун агентуудыг микросервис архитектурт нэвтрүүлэхэд тулгарах асуудлуудыг тодорхойлж, ижил төстэй системүүдтэй (Inngest, Temporal, Camunda) харьцуулан судлах
5. Apache Kafka болон Apache Flink ашиглан үйл явдлаар удирдагдах агентын микросервис архитектурыг санал болгох
6. Санал болгосон зохиомжийг туршилтаар хэрэгжүүлэх: Зохион байгуулагч агент, Мэдлэгийн агент, Хөрөнгө оруулалтын агент, мэдээний агент, PyFlink төлөвлөгч зэрэг системийн бүрэлдэхүүн хэсгүүдийг хөгжүүлэх
7. МХБ-ийн бодит өгөгдөл (компанийн мэдээлэл, арилжааны түүх) ашиглан демо систем бүтээж, Next.js 14 фронтенд, API Gateway болон агентүүдийг холбох

8. Системийг Docker орчинд deploy хийж, PostgreSQL, Kafka, Redis зэрэг программ хангамжуудыг тохируулах
9. Бүрэн хэрэгжүүлсэн системийг тест хийж, өргөжих чадвар, найдвартай байдал, агентуудын хамтын ажиллагааг үнэлэх

1. ОНОЛЫН ХЭСЭГ

1.1 Хиймэл оюуны инженерчлэл гэж юу вэ

Хиймэл оюуны инженерчлэл гэдэг нь бэлэн бэлтгэгдсэн суурь модел дээр аппликейшн хөгжүүлэх үйл явцийг хэлнэ. Энэ нь уламжлалт машин сургалтын инженерчлэл ялгаатай байдаг. Хэрэв уламжлалт машин сургалтын инженерчлэл нь модел хөгжүүлэхэд чиглэсэн бол, хиймэл оюуны инженерчлэл нь бэлтгэгдсэн моделийг программ хангамжид интеграц хийхэд чиглэсэн байдаг. [1].



Зураг 1.1: Хиймэл оюуны инженерчлэл ба машин сургалтын инженерчлэл

OpenAI, Anthropic зэрэг компаниудын гаргасан хүчирхэг суурь моделийн 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 модел ашиглан хэрэглээнд нэвтрүүлэх нь илүү боломжтой болов. Улмаас хиймэл оюуны инженерүүд программ хангамжийг хүртээмжтэй моделүүд 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 моделийн хөгжил

Хэл моделүүдээс том хэлний модел болон суурь модел 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 моделийн үндэс

Хэл модел 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 модел нь "машин" биш, харин "цэнхэр" гэсэн үгийг таамаглах ёстой. [1]

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

Хэл моделд хоёр үндсэн төрөл байдаг. Эхний төрөл нь далдлагдсан хэлний моделууд бөгөөд эдгээр нь өгүүлбэр доторх далдлагдсан үгсийг таамаглах замаар сурдаг. BERT нь энэ төрлийн алдартай жишээ юм. Хоёр дахь төрөл нь авторегрессив хэл моделууд 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

Хэл моделийн хамгийн чухал давуу тал нь өөрийгөө удирдсан сургалтыг ашиглах чадвар юм. 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 моделийг номнуудаас, блог нийтлэлээс, өгүүллүүд, Reddit-ийн сэтгэгдэл зэргээс ашиглан сургаж болно. Энэ нь асар их сургалтын өгөгдөл бүрдүүлэх боломжийг олгож, хэл моделийг том хэлний модел буюу LLM болтол өргөжүүлэх боломжтой болгосон.

1.2.3 Том хэлний моделоос суурь модел руу

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

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

Суурь моделууд нь LLM-ээс цааш өргөжсөн ойлголт юм. Эдгээр нь зөвхөн текст биш, зураг, аудио, видео зэрэг олон төрлийн өгөгдөл боловсруулж чаддаг том мульти модал моделууд юм. Суурь моделийн гол онцлог нь тодорхой үүрэгтэй моделээс цаашлаад ерөнхий зориулалтын модел 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. 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 моделийн сургалт

Суурь моделийг бэлтгэх нь хоёр үндсэн үе шаттай:

1.4.1 Урьдчилан сургалт

Урьдчилан сургалт нь өөрийгөө удирдсан сургалт ашиглан их хэмжээний өгөгдөл дээр моделийг сургах үйл явц юм. Энэ үе шатанд модел 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 модел 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 модел 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 модел дээр өөр өөр ажиллах зарчимтай байна. Сургалтан дээр суурь моделийн ойролцоогоор тал хувийг англи хэл эзлэдэг бол орос, герман хэл 10 хувийг эзлэж байна. GPT-4 суурь моделийн 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 моделийн хэлний хязгаарлалтыг давахын тулд өөрсдийн хэл дээр суурь модел хөгжүүлж байна. Жишээлбэл, хятадын "Llama-Chinese", францийн "Croissant-LLM", Вьетнамын "PhoGPT" гэх зэрэг. Монгол улсын хувьд "Чимэгэ систем" нь монгол хэл дээр суурь модел 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 моделийг 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 моделийг нарийвчлан сургаж, зөвхөн өгүүлбэрийн гүйцээлт биш харин харилцан ярианы горимд оновчтой болгоно. Хоёр дахь үе шат нь сонголтын нарийвчилсан сургалт юм. Энэ үе шатанд моделийг 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

Модел нь гаралтаа sampling процессоор бүтээдэг. Sampling нь хиймэл оюуны гаралтын магадлалыг шууд нөлөөлдөг. [1] Температур нь моделийн бүтээлч чанарыг удирдах гол параметр юм. Температур өндөр байх тусам модел илүү бүтээлч, гэнэтийн хариулт өгдөг бол температур бага байх тусам илүү таамагладаг, 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 уран

зохиолд ялангуяа 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, урт хариулт, богино хариултын моделиг тодорхойлж болдог. Иймээс хэрэглээнээс хамаарч sampling стратеги сонгох нь чухал. Жишээлбэл тийм, үгүй сонголттой асуулт хариулт гаргах, урт тэмдэгт мөртэй хариулт гаргах гэх мэт байж болно.

1.4.4 Моделийн үр дүнг хэмжих

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

Сүүлийн жилүүдэд бага параметртэй модел нь өмнөх үеийн их параметртэй моделээс илүү чадалтай байна. Жишээлбэл, 2024 оны Llama 3-8B модел нь 2023 оны Llama 2-70B моделээс ч илүү сайн үр дүнг MMLU benchmark дээр харуулжээ. Энэ нь зөвхөн моделийн хэмжээ биш, сургалтын аргууд болон өгөгдлийн чанар хамгийн чухал болохыг харуулж байна.

Үнэлгээний хувьд гурван гол асуудал тулгардаг. Нэг дүгээрт, суурь моделуудыг 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, модел нь ижил эсвэл бага зэрэг өөр асуулт асуухад маш өөр хариулт өгч болох тогтворгүй байдал байдаг. Үүнээс хиймэл оюуны суурь моделийн хариулт нь магадлалаас үүсдгийг ажиглаж болно. Гуравдугаарт, модел нь баримт дээр үндэслээгүй буруу хариулт буюу төөрөгдөл үүсгэж болдог.

1.5 Зааврын инженерчлэл

Зааврын инженерчлэл гэдэг нь моделоос хүссэн үр дүнг гаргуулахын тулд зааврыг бичих үйл явц юм. Энэ нь моделийн жинг өөрчлөхгүйгээр зан үйлийг удирдах хамгийн хялбар бөгөөд түгээмэл моделийн дасан зохицох арга юм.

1.5.1 Зааврын бичих шилдэг арга барил

Заавруудыг хэрэглээнд тохирсон стратегиудын дагуу бичих нь илүү сайн үр дүн өгдөг. [7] OpenAI-ийн санал болгож буй заавар бичих шилдэг арга барил нь эхлээд юу хийлгэхээ хоёрдмол утгагүй байдлаар тодорхой тайлбарлах хэрэгтэй. Дараа нь моделоор тодорхой дүрд тоглуулж болно. Жишээ нь "Та том компанид 20 жил ажилласан туршлагатай программист. Кодыг шалгаад сайжруулж өг" гэх мэт. Anthropic-ийн зөвлөмжөөр зааварт 500 хуудас бүхий номын урттай тэмдэгт мөр багтаж чадах тул зааварт урт жишээ өгснөөр хариултын формат болон хариултын хоёрдмол утгыг багасгадаг. Цар хүрээ, агуулгыг мэдээлүүлснээр төөрөгдлийг багасгадаг. Хэрэв модел шаардлагатай мэдээллээр хангагдаагүй бол өөрийн дотоод мэдлэгтээ найдах бөгөөд энэ нь найдваргүй байж болдог. Түүнчлэн нарийн төвөгтэй даалгавруудыг хялбар дэд даалгавруудад 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 моделийн зөвшөөрөөгүй үйлдэл хийлгэх оролдлого юм. Нөгөө нь моделийн сургалтын өгөгдөл эсвэл контекстын мэдээллийг задруулах оролдлого юм. [1]

Иймээс хиймэл оюуны агент эсвэл модел угсарч буй тохиолдолд оролт болон гаралтыг үнэлэж хамгаалах функц нэвтрүүлэх нь мэдээллийн аюулгүй байдлыг хангадаг.

1.6 Хайлтаар нэмэгдүүлсэн үүсгэлт (RAG)

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

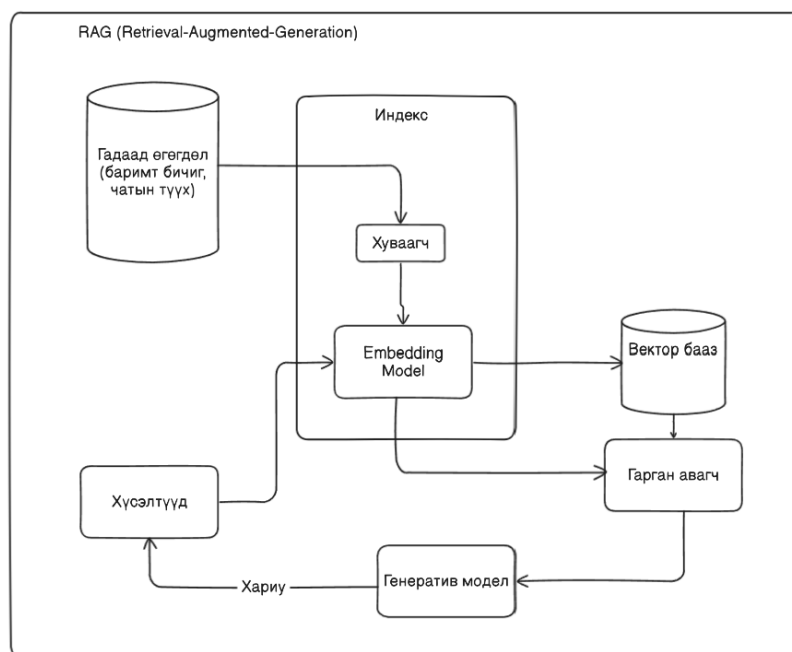
Хэдийгээр моделийн контекстийн урт тогтмол нэмэгдэж байгаа ч RAG-ийн ач холбогдол алдагдахгүй байна. Зарим аппликейшнд өгөгдлийн хэмжээ байнга өсч байдаг. Иймээс RAG удааширж магадгүй тул үүнийг сайтар үнэлж, тасралтгүй

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 модел 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

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.

Нэр томьёо суурилсан хайлт

Энэ арга нь түлхүүр үгээр баримт хайдаг. Энэ арга Google, Bing зэрэг хөгчийн 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 векторчилсон хэсгийн төлөөлөл болгон хувиргагдаж, векторын өгөгдлийн санд хадгалагдана. Асуулт ирэх үед түүний векторчилсон төлөөлөлтэй 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 хэсэглэх нь чухал. Тогтмол урттай хэсэглэх, өгүүлбэр догол мөрөөр хэсэглэх, утга зүйгээр хэсэглэх зэрэг аргууд байдаг. Анхны хайлтын үр дүнг дахин эрэмбэлэн илүү нарийвчлалтай болгох арга байдаг. Асуултыг дахин найруулж илүү сайн хайлт хийх нь бас үр дүнтэй. Нэр томьёо болон векторчилсон хэсгийн төлөөлөл суурилсан хайлтыг хослуулсан арга ашиглаж болно. Хэсэг бүрийг metadata, түлхүүр үг, холбогдох асуултуудаар баяжуулах нь хайлтын чанарыг сайжруулдаг.

1.7 Хиймэл оюуны агентууд

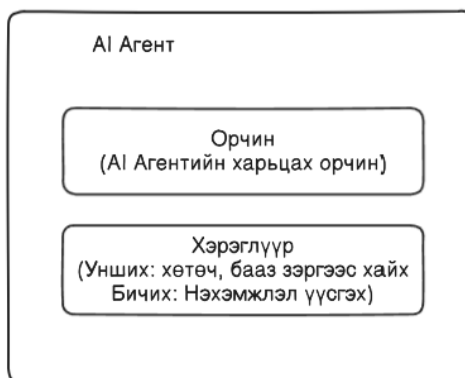
1.7.1 Агент

Агент гэдэг нь өөрийн орчныг мэдрэх, түүн дээр үйлдэл хийх чадвартай систем юм. [1]. Хиймэл оюунаар дэмжигдсэн агентууд нь суурь моделийн 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 гурван 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. Даалгавар нь хэрэглэгчээс өгөгдсөн ажил юм.



Зураг 1.3: Агентын бүрэлдэхүүн хэсгүүд

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.

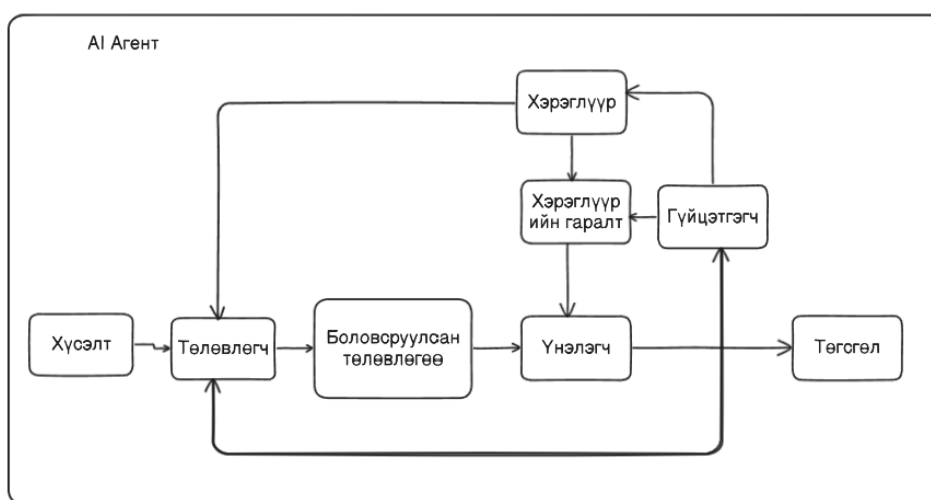
Мэдлэг нэмэгдүүлэх хэрэглүүрүүд нь хайлтаар нэмэгдүүлсэн үүсгэлт систем буюу RAG-ын бүрэлдэхүүн хэсгүүд юм. Үүнд текст хайгч, зураг хайгч, SQL гүйцэтгэгч, интернэт хайлтын програмчлалын интерфэйс, дотоод хайлтын системүүд зэрэг багтана.

Чадавх өргөтгөх хэрэглүүрүүд нь агентын үндсэн чадварыг өргөжүүлдэг. Хиймэл оюун моделууд 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 кодчиллын 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 зааврын довтолгооноос болгоомжлох хэрэгтэй.

1.7.4 Төлөвлөлт

Төлөвлөлт нь агентын гол үүрэг бөгөөд олон үе шаттай. Эхний үе шат нь төлөвлөгөө үүсгэх явдал юм. Даалгаврыг гүйцэтгэх дараалсан үйлдлүүдийн төлөвлөгөө гаргах энэ үйл явцыг даалгаврыг дэд процесс гэж нэрлэнэ. 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 хийх бөгөөд энэ нь ихэвчлэн функц дуудах үйлдэл хийнэ. Эцсийн үе шат нь үр дүнг үнэлэх явдал юм. Үйлдлийн үр дүнг хүлээн авсны дараа зорилго биелсэн эсэхийг тодорхойлж, алдааг тодорхойлж засна.



Зураг 1.4: Агентын төлөвлөлт

1.7.5 Суурь моделууд төлөвлөгч болж чадах уу

Зарим судлаачид том хэлний модел нь төлөвлөгч болж чадахгүй гэж үздэг. [16]. Учир нь төлөвлөлт нь үндсэндээ хайлтын асуудал бөгөөд авторегрессив буюу магадлалт суурилсан модел нь зөвхөн нэг чиглэлийн үйлдэл үүсгэж чаддаг гэж үздэг. Гэвч бодит байдал дээр модел дахин эхлэж өөр зам сонгож чаддаг тул сургалт сайн хийснээр сайн төлөвлөгөө гарч болно.

Төлөвлөлтийг сайжруулах олон арга байдаг. Илүү сайн системийн заавар бичиж, жишээ олноор өгөх нь чухал. Хэрэглүүрүүдийн тайлбарыг илүү сайн бичих хэрэгтэй.

Функцүүдийг хялбарчлах, задлах нь төлөвлөлтийг хөнгөвчилдөг. Илүү хүчтэй модел ашиглах нь илүү сайн төлөвлөгөө гаргах боломжийг олгоно. Төлөвлөлтөд зориулж моделийг нарийвчлан сургах нь бас үр дүнтэй. Зарим үнэлгээний шинжилгээгээр оновчтой хэрэглүүр өгөх нь нарийвчилсан сургалт хийснээс илүү үр дүнтэй бас хямд гэж үзжээ.

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 модел өөртэйгөө ярилцаж алдааг илрүүлдэг. Хоёр дахь арга нь тусдаа үнэлэгч ашиглах явдал юм. Тусдаа модел эсвэл функц үр дүнд оноо өгдөг.

1.7.7 Агентын санах ой

Хиймэл оюун модел нь гурван санах ойн механизмтай. [1]. Дотоод мэдлэг нь модел өөрөө дотоод мэдээлэлтэй байдаг. Энэ нь сургалтын өгөгдлөөс олж авсан мэдлэг бөгөөд моделийг шинэчлэхгүй бол өөрчлөгдөхгүй. Богино хугацааны санах ой нь моделийн 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.

Эргэцүүлэн бодох

Эргэцүүлэн бодох нь агентуудад өөрсдийн шийдвэрийг үнэлж, үйлдэл хийх эсвэл эцсийн хариулт өгөхөөсөө өмнө гаралтаа сайжруулах боломжийг олгодог. Энэ чадвар нь агентуудад алдаагаа олж засах, дүгнэлтээ боловсронгуй болгох, илүү өндөр чанартай үр дүн гаргах боломжийг олгоно.

Жишээлбэл, код бичиж байгаа агент нь эхлээд код үүсгээд, дараа нь өөрөө тухайн кодыг шалгаж, алдаа олж, сайжруулалт хийснийхээ дараа хэрэглэгчид хүргэнэ. Энэ нь эцсийн үр дүнгийн чанарыг мэдэгдэхүйц сайжруулдаг.

Төлөвлөлт

Төлөвлөлтийн чадвартай агентууд нь өндөр түвшний зорилгуудыг үйлдэл хийх боломжтой алхмуудад задалж, даалгавруудыг логик дарааллаар зохион байгуулж чаддаг. Энэ зохиомж нь олон алхам бүхий асуудлыг шийдэх эсвэл хамааралтай ажлын урсгалыг удирдахад чухал юм.

Жишээлбэл, аялал төлөвлөх агент нь эхлээд нислэг хайж, дараа нь зочид буудал захиалж, үзвэр сервисийн цэгүүдийг судалж, эцэст нь өдөр бүрийн маршрут үүсгэх төлөвлөгөө гаргаж болно.

1.7.8 Яагаад олон агентын зохиомжийн хэрэгтэй вэ?

Агент нь тодорхой даалгаврын хүрээнд ажил гүйцэтгэх чадвартай байдаг. Жинхэнэ бизнесд үйл ажиллагаа нь олон дэд процессд хуваагдаж болдог шиг агентийн хэрэглээ мөн адил нарийн ажиллагаатай байж болдог. Тиймээс олон агентийн зохиомж нь агентуудад нарийн төвөгтэй асуудлыг шийдэх, хувьсах орчинд дасан зохицох, үр дүнтэй хамтран ажиллах боломжийг олгодог. Жишээлбэл өөрсдийн сургасан моделийг тооцоолоход үнэтэй дэд процессд үлдээгээд, хямдхан GPT-2 зэрэг моделээр өөр үүрэгтэй үйлдлүүдэд үлдээж болно. Иймээс мульти-агент байх нь практикт тохиромжтой түгээмэл арга зам юм.

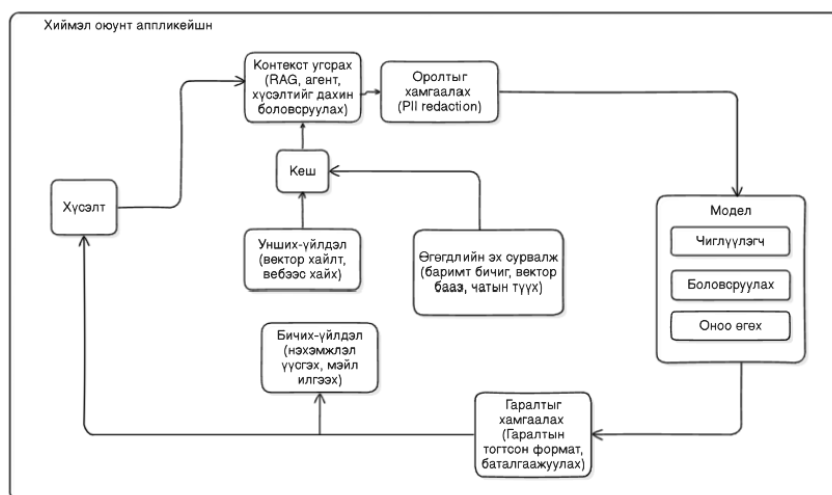
Олон агентын хамтын ажиллагаа

Олон агентын системүүд нь асуудлыг шийдэхэд модуляр арга барил ашигладаг. Иймээс тодорхой даалгавруудыг мэргэшсэн агентуудад хуваарилдаг. Энэ арга нь уян хатан байдлыг санал болгохоос гадна зардлыг хэмнэнэ. Мөн үр ашигтай байдлыг сайжруулахын тулд даалгаварт агентуудад жижиг хэлний моделууд ашиглаж болно. Жишээлбэл хэрэглэгчийн хүсэлтийг танихад.

Модульчлагдсан зохиомж нь агентуудын цар хүрээ дотор тусгай даалгаврууддаа чиглүүлэх замаар агент тус бүрийн нарийн төвөгтэй байдлыг багасгадаг. Хамтран ажиллахын тулд мэргэшсэн агентууд мэдээлэл солилцож, хариуцлагыг хуваарилж, нарийн төвөгтэй сорилтуудыг илүү үр дүнтэй шийдэхийн тулд үйлдлүүдээ зохицуулдаг.

Уламжлалт системийн зохиомжтай адилаар, асуудлыг модульчлагдсан бүрэлдэхүүн хэсгүүдэд задлах нь тэднийг засварлах, өргөжүүлэх, дасан зохицуулахад илүү хялбар болгодог.

1.8 Жишээ хиймэл оюунт программ хангамжийн зохиомж



Зураг 1.5: Хиймэл оюунт аппликейшн

Суурь моделууд ашиглан аппликейшн хөгжүүлэх нь уламжлалт машин сургалтын инженерчлэлээс гурван талаараа ялгаатай байдаг. Эхнийх нь моделийн дасан зохицуулалт юм. Өөрөө модел сургахын оронд бусдын сургасан моделийг ашиглаж болдог. Иймээс уг моделийг дасан зохицуулах нь чухал болсон. Хоёр дахь нь оновчлол юм. Суурь модел нь илүү их тооцоолол шаарддаг, хоцрогдол өндөр байдаг учраас үр ашигтай дасан зохицуулалт хийснээр бага хүчтэй компьютерээр, бага хоцрогдлоор гүйцэтгүүлж болно. Иймээс бэлэн суурь моделийн програмчлалын интерфэйс ашиглах уу, эсвэл өөрөө агентээ серверт байршуулах уу гэдгийг шийдэж болно. Гурав дахь нь нээлттэй гаралт юм. Моделууд нээлттэй төгсгөлтэй гаралт үүсгэдэг тул үнэлгээ хийхэд илүү бэрхшээлтэй байдаг. Дундах процессийг мэдэж болдоггүй.

Моделийг дасан зохицуулах хоёр гол арга байдаг. Заавар суурилсан техникүүд нь моделийн жин өөрчлөхгүйгээр зааварчилгаа, контекст өгч дасан зохицуулдаг. Хялбар, цөөн өгөгдөл шаарддаг. Зааврын инженерчлэл энд хамаарна. Нарийвчилсан сургалт нь

моделийн жинг өөрчилж дасан зохицуулдаг. Илүү нарийн төвөгтэй, илүү их өгөгдөл шаарддаг. Гэвч чанар, хоцрогдол, өртгийг мэдэгдэхүйц сайжруулж чадна. Хэрэглүүр өгөх нь уг даалгаврыг маш оновчтой, хоцрогдол багатайгаар, зардлын өртгийг мэдэгдэхүйц сайжруулж чадна.

1.9 Бүлгийн дүгнэлт

Хиймэл оюуны инженерчлэл нь уламжлалт машин сургалтын инженерчлэлээс өөр чиглэл бөгөөд бэлтгэгдсэн суурь модел дээр програм хангамж хөгжүүлэхэд чиглэсэн шинэ салбар юм. [1]. OpenAI, Anthropic зэрэг компаниудын хүчирхэг суурь моделийн нэвтрүүлэлт нь хиймэл оюуныг програм хангамжид интеграц хийхийг илүү боломжтой болгож, салбарын хурдацтай өсөлтийг бий болгосон.

Суурь моделийн хөгжлийн замнал нь энгийн хэл моделөөс том хэлний модел, улмаар суурь модел руу шилжих үйл явц байсан. [1]. Өөрийгөө удирдсан сургалт нь асар их хэмжээний өгөгдлөөр модел сургах боломжийг нээсэн. 2017 оны Transformer архитектур нь хэл моделийн чадамжийг шинэ түвшинд ахиулсан. Урьдчилан сургалт болон дараах сургалтын хослол нь моделд ерөнхий мэдлэгийг өгөөд, хэрэглэгчийн хүсэлтэд тохируулах боломжийг бүрдүүлсэн. Мөн Sampling стратеги, температур, top-k, top-p зэрэг параметрууд нь моделийн гаралтын чанар, олон янз байдлыг магадлал дээр удирддаг.

Зааврын инженерчлэл нь модел дасан зохицуулах хамгийн хялбар арга. Зааврыг тодорхой, нарийвчлалтай бичих, модельд дүр олгох, урт жишээ өгөх, контекст хангах зэрэг техникүүд нь моделийн ажиллагааг мэдэгдэхүйц сайжруулдаг. Иймээс машин сургалт хийхгүйгээр программ хангамжид хиймэл оюун интеграц хийж болно. Гэвч хортой зааврын довтолгоо, мэдээллийг задруулах оролдлогоос хамгаалах нь чухал асуудал бөгөөд оролт болон гаралтыг үнэлэх функц нэвтрүүлэх шаардлагатай.

Хайлтаар нэмэгдүүлсэн үүсгэлт буюу RAG нь моделийн дотоод мэдлэгийн хязгаарлалтыг даван, гадаад эх сурвалжаас мэдээлэл авч хариултын найдвартай байдлыг нэмэгдүүлдэг. Хэдийгээр моделийн контекстын урт нь хайлтад урвуугаар нөлөөлдөг ч RAG-ийн ач

холбогдол алдагдахгүй байна. Урт контекст нь үргэлж үр ашигтай ашиглагдахгүй бөгөөд нэмэлт өртөг, хоцрогдол авчирдаг. RAG систем нь хайгч болон үүсгэгч гэсэн хоёр гол бүрэлдэхүүнтэй. Хайлтын чанар нь системийн амжилтын түлхүүр болох тул нэр томъёо суурилсан хайлт, embedding суурилсан хайлт, эсвэл хоёулаа нэгтгэсэн аргыг ашиглан оновчтой байх хэрэгтэй.

Хиймэл оюун агент нь өөрийн орчныг мэдрэх, түүн дээр үйлдэл хийх чадвартай систем юм. Агентын гурван гол бүрэлдэхүүн нь орчин, үйлдэл, даалгавар болох нь тодорхой болсон. Хэрэглүүрүүд нь агентын чадавхийг өргөтгөх чухал элемент бөгөөд мэдлэг нэмэгдүүлэх, чадавх өргөтгөх, бичих үйлдлийн гэсэн гурван төрөлд хуваагдана. Төлөвлөлт нь агентын гол үүрэг болох нь илэрхий. Төлөвлөгөө үүсгэх, эргэцүүлэн бодох, гүйцэтгэх, үр дүнг үнэлэх дөрвөн үе шат нь агентыг уян хатан, найдвартай болгодог.

Олон агентын зохиомж нь нарийн төвөгтэй бизнес процессуудыг шийдэхэд илүү тохиромжтой байдаг. Агент бүр цар хүрээндээ мэргэшсэн даалгавартай байх нь системийг модульчлагдсан, өргөжүүлэх боломжтой болгодог. Агентуудын хамтын ажиллагаа, мэдээлэл солилцох чадвар нь ганц агентаас илүү хүчирхэг шийдлийг бий болгоно.

Эцэст нь хиймэл оюуны инженерчлэл нь зөвхөн машин сургалтын технологийн асуудал биш, харин цогц системийн архитектур, систем зохион байгуулалт, хэрэглэгчийн туршлагыг хослуулсан цогц салбар юм. Суурь модел, зааврын инженерчлэл, RAG систем, агентын архитектур зэрэг онолууд нь дараагийн бүлгүүдэд гарах болно.

2. МИКРОСЕРВИС АРХИТЕКТУР

2.1 Монолитоос микросервис рүү

2.1.1 *Монолитын эрин үе*

Вэб аппликейшн хөгжүүлэлтийн эхэн үед бүх зүйлийг монолит хэлбэрээр бүтээдэг байсан. Бүх бизнес логик, өгөгдлийн логикууд нэг том, нэгдсэн кодын санд амьдардаг байв. Программ хангамжийн өөгжүүлэгчид 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 энэхүү кодын санд орсноор энэ нь шинэчлэлтийг удаашруулж, унахад эрсдэлтэй болгодог. Өөр өөр функц дээр ажиллаж байгаа багууд байнга бие биенийхээ кодоод хамаарал бүхий харилцаагаар холбогдож, алдаа нэмж, хөгжлийг удаашруулж, алдаа гарах эрсдэлийг нэмэгдүүлсээр ирсэн.

Монолит архитектураас эхэлсэн компаниуд эцэст нь эдгээр асуудлууд тулгарсан. Хурдан үйл ажиллагаагаа явуулахад багууд системийг шинэчлэх, турших хүртэл хүлээх шаардлагатай байв. Энэ нь компаний бизнесийн үйл ажиллагаанд үндсэн саад бэрхшээл болов.

2.1.3 *Микросервис рүү шилжих*

Эдгээр хязгаарлалтаас ангижрахын тулд компаниуд микросервис архитектур руу шилжүүлсэн. [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, зохицуулалтын ачааллыг бууруулж, инновацийг хурдасгасан.

2.1.4 Микросервисийн тодорхойлолт

Микросервис архитектур нь програм хангамжийг хөгжүүлэх арга бөгөөд аппликейшныг жижиг, бие даасан сервисүүдэд хувааж 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].

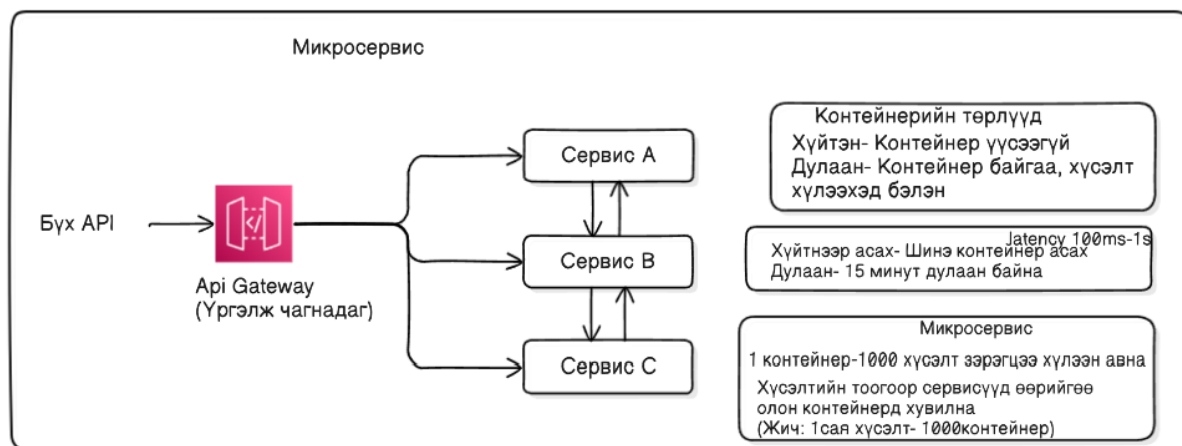
Өгөгдлийн тогтмол байдлын асуудал нь сервис бүр өөрийн өгөгдлийн сантай байдгаас үүсдэг. Олон сервисийн өгөгдлийн нийцтэй, уялдаатай байдлыг хангах нь уламжлалт транзакцийн удирдлагаар шийдэгдэхгүй асуудал болдог. Сүлжээний хоцрогдлын талаас авч үзвэл сервисүүд хоорондоо сүлжээгээр харилцдаг учир нэмэлт 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].

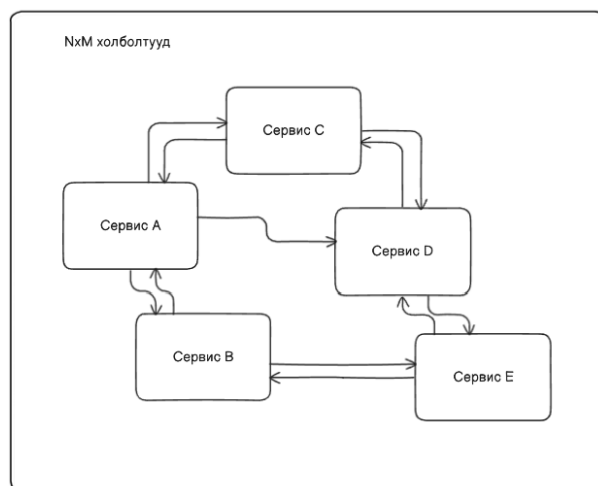
2.4.1 Синхрон харилцаа

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 сервис 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.



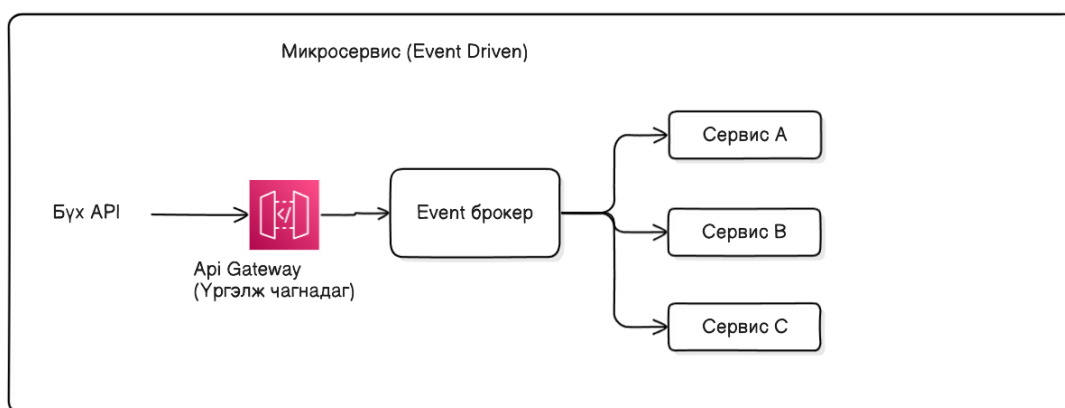
Зураг 2.1: Синхрон микросервис

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

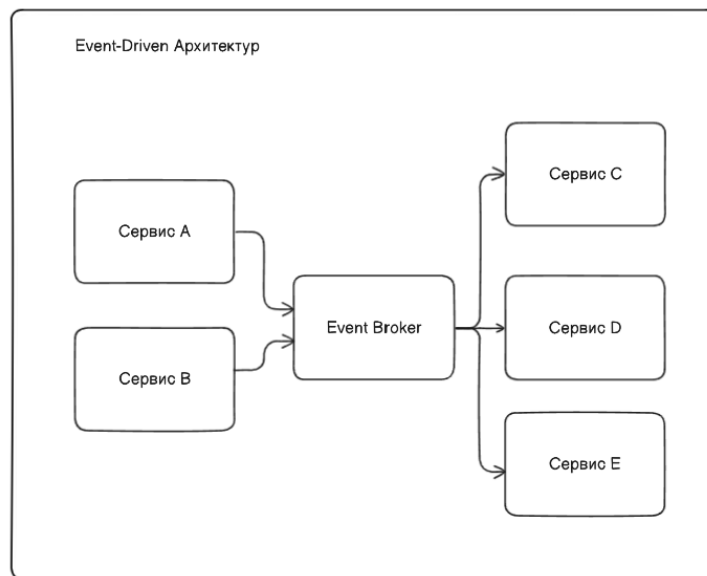


Зураг 2.3: Асинхрон микросервис

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 архитектурт 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 групп буюу consumer group нь олон consumer нэг баг болон зохион байгуулагдаж ачааллыг хуваариалан 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, модел 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.

Flink ба хиймэл оюун

Flink нь том хэлний моделтэй ажиллах чадвартай. [17] Flink нь өгөгдлийг авч, том хэлний модел 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 модел 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 рекурсив байдлаар ажиллаж чаддаг. Энэхүү зохион байгуулалт нь урсгал боловсруулалтын давуу талыг хиймэл оюун агенттай нэгтгэж чаддаг.

Flink болон RAG

Төөрөгдлийг багасгахын тулд том хэлний моделийг 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 моделийн 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 моделийн мэдлэгийг тасралтгүй шинэчилж байх боломжийг олгодог.

2.5.5 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.

2.6 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 моделтэй 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. АСУУДЛЫН ТОДОРХОЙЛОЛТ БА ШИЙДЭЛ

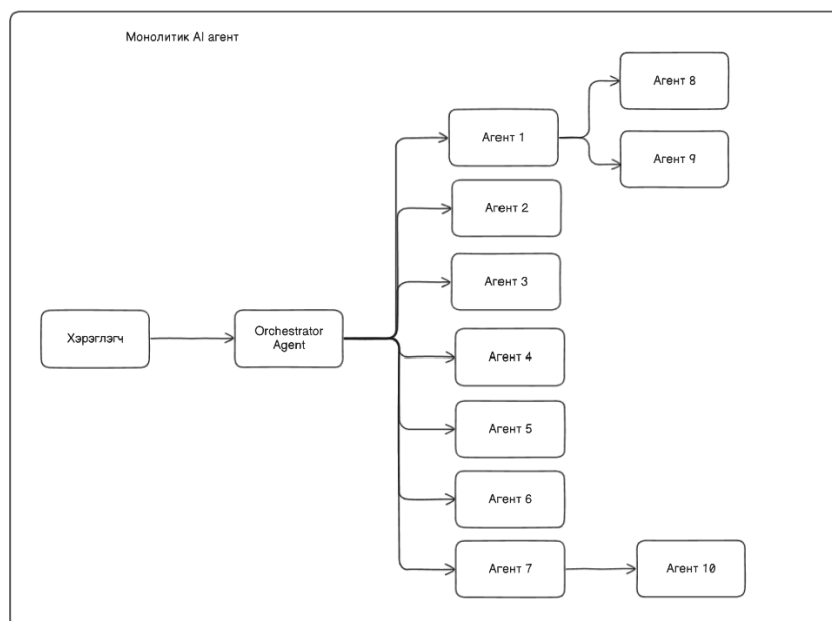
Энэхүү бүлэгт олон агентын системийг монолит, микросервис архитектураар нэвтрүүлэхэд тулгарах давуу, сул тал, асуудлуудыг харьцуулан тодорхойлж, EDA ашиглан ухаалаг байдлаар бие даан ажиллах чадвартай олон агент системуудын архитектурын зохиомжийн шийдлийг санал болгоно.

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 модел 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

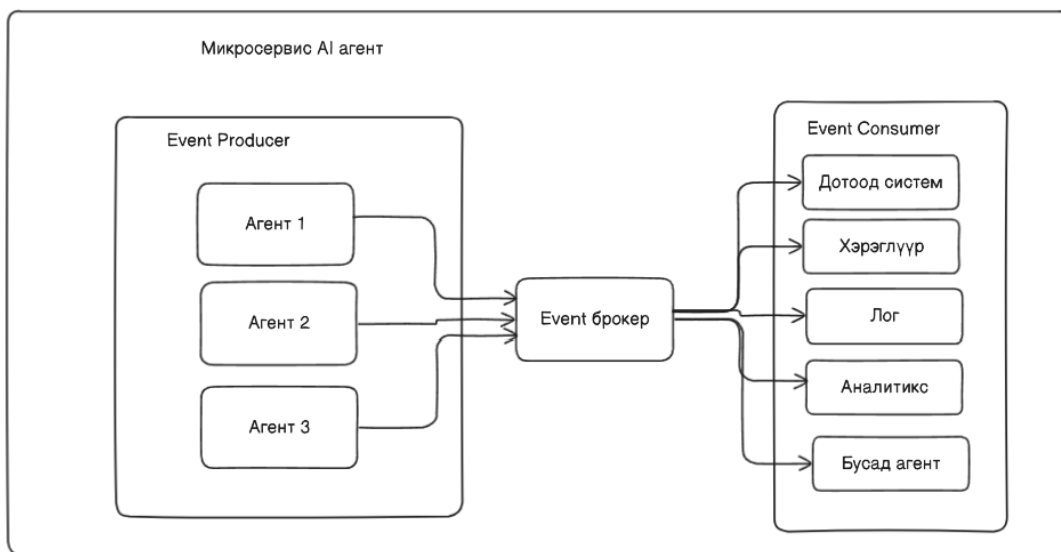
Монолит агентын системийн дараах сул талуудтай байна. Агентууд бие биенээсээ шууд хамааралтай байдаг учир нэг агентын өөрчлөлт нь бусад агентуудад нөлөөлдөг. Бүх агентууд нэгэн зэрэг өргөжих ёстой бөгөөд тусдаа өргөжүүлэх боломжгүй тул өргөжүүлэх хүндрэлтэй байдал үүсдэг. Нэг агентыг шинэчлэхэд бүх системийг дахин суурилуулах шаардлагатай тул хөгжүүлэлтийн багууд бие биенийгээ хүлээж бизнесийн үйл ажиллагаа удаашрах эрсдэлтэй. Бүх агентууд нэг серверийн нөөцийг хуваалцдаг, иймээс олон хүсэлт даахад бэрхшээлтэй болно. Нэг кодын сан дээр ажиллах нь зөрчил, удаашралыг үүсгэдэг зэрэг эрсдэл байдаг.

Нэг агент унах нь бүх системийг доголдох магадлалтай. Жишээлбэл, Knowledge Agent унавал Orchestrator түүнээс хүлээж байдаг учир бусад хүсэлтүүд боловсруулагдахгүй. Энэ нь монолит агентын хамгийн том эрсдэл болох дамжин унах алдаа юм.

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 брокерийн мессежээр харилцдаг. Энэ нь хамааралыг $N+M$ болгон бууруулж, агентуудыг бүрэн салангид болгодог.



Зураг 3.2: Үйл явдлаар удирдагдах олон агентын систем

Үйл явдлаар удирдагдах агентын систем нь гурван гол бүрэлдэхүүнтэй. Эвент үүсгэгчүүд нь өөр өөр зориулалттай агентууд (Төлөвлөгч, чиглүүлэгч, мэдээ, төлбөр тооцооны гэх мэт) үйл явдал үүсгэж эвент брокер руу илгээдэг. Apache Kafka, RabbitMQ зэрэг эвент брокер нь агентуудын хоорондын төв мэдрэгч систем болж үйл явдлыг хадгалж агентууд өөр өөрдийгөө рекурсив байдлаар дуудах боломжийг нээдэг. Эвент хүлээн авагч хэсэг нь агентууд өөрсдөө ба брокерийн гаргаж буй үйл явдлыг хүлээн авч боловсруулна. Үүнд агентын хэрэглүүрүүд (жишээ нь цаг агаарын мэдээ авах, нэхэмжлэл үүсгэх..), систем интеграци, лог болон шинжилгээ, бусад агент руу чиглүүлэх зэрэг багтдаг.

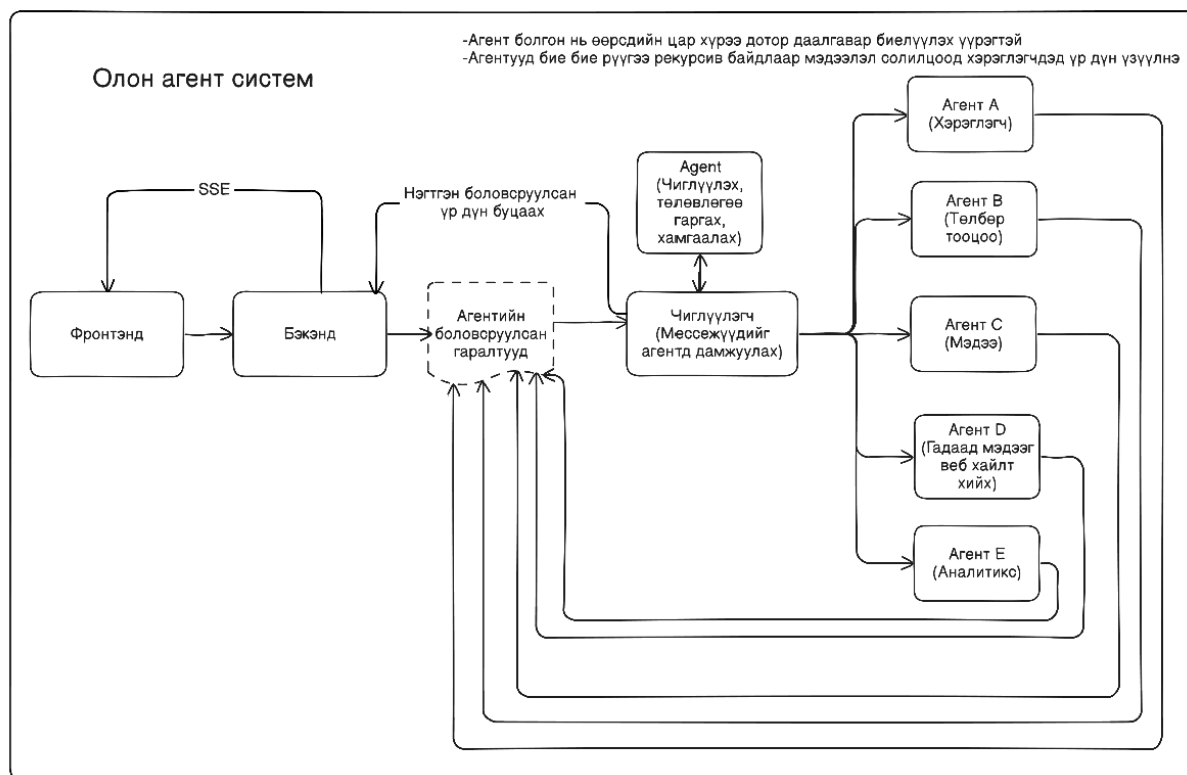
Энэхүү архитектур нь монолит системийн бүх сул талыг шийддэг. Салангид байдлын хувьд агентууд эвент брокероор харилцах тул бие биенээсээ хараат бус байдаг. Нэг агент унах нь бусдад шууд нөлөөлөхгүй. Хэрэглүүр, системтэй холбогдсон байдал нь хуучраагүй бодит цагийн өгөгдөл авах зэрэг боломж нээдэг. Асинхрон 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

Энэ судалгааны ажлын санал болгож буй микросервист суурилсан хиймэл оюуны агентууд дараах байдлаар ажиллана. Хэрэглэгч фронтенд дээр хүсэлт илгээнэ. API Gateway хүсэлтийг хүлээн авч Kafka сэдэв рүү үйл явдэл болгон илгээнэ. Зохион байгуулагч агент үйл явдлыг хүлээн авч суурь модел эсвэл жижиг GPT-2 зэрэг моделоор хүсэлтийг ангилна. Ангилласны дараа олон агентуудын мэдээллийг агуулж буй RAG-аас аль агент руу чиглүүлэхийг заана. Мөн Apache Flink ашиглан нарийн төвөгтэй төлөвлөлт хийх, хиймэл оюуны боловсруулсан мэдээллийг нэгтгэж болдог. Зохион байгуулагч агент нь эцэст нь уг нэгтгэсэн хариуг хянаж тохирох агент, эсвэл хэрэглэгч рүү хариу илгээж болно.

Хортой зааврыг урьдчилсан сэргийлэхийн тулд уг хүсэлтийг боловсруулж ангилаад хэрэглэгчдэд хүргэхээс өмнө хааж болно. Мөн эцсийн боловсруулсан гаралтыг бас хянаад хааж болдог.



Зураг 3.3: Бодит хэрэгжүүлсэн архитектур: Kafka-Flink суурилсан олон агентын систем

Өөр өөр зориулттай агентууд өөрсдийн үүргийг гүйцэтгэдэг. Knowledge Agent буюу мэдлэгийн агент RAG нь бусад агент, чадавхын мэдээлэл агуулах ба уг агентийг ашиглан хүсэлтийг хаашаа чиглүүлж болох талаар хайж шийдвэр гаргана. [1]. Мэдээллийн агент нь өөрсдийн хандах боломжтой орчноос хэрэглүүрийн тусламжтай мэдээлэл цуглуулна. Төлбөр тооцооны агент нь гүйлгээ боловсруулах зэрэг үйлдэл хийнэ. Хэрэглэгчийн агент нь хэрэглэгчийн мэдээлэлтэй ажилладаг. Агентууд үр дүнгээ Kafka сэдэв рүү буцааж илгээнэ.

Зарим тохиолдолд агентууд рекурсив байдлаар зохион байгуулагч агент руу дахин хандаж нэмэлт мэдээлэл авч болно. Энэ нь нарийн төвөгтэй олон алхамт даалгаврыг шийдэх боломжийг олгодог. Жишээлбэл хэрэглэгчийн мэдээллийг авч, түүний төлбөр тооцооны мэдээг өөр агентаас авч нэгтгэж болно. Эцэст нь API Gateway нь хариултыг хүлээн авч хэрэглэгчид SSE (Server-Sent Events) ашиглан бодит цагийн stream хэлбэрээр хүргэнэ.

Энэхүү архитектур нь дараах давуу талуудтай. Агент бүр бие даан параллелаар ажиллаж, мянга мянган хүсэлтийг боловсруулна. Шинэ агент нэмэх нь бусад агентын кодыг өөрчлөхгүй. Бодит цагийн шийдвэр гаргалтын хувьд хатуу кодлогдсон процессийн урсгал биш, хиймэл оюун динамикаар дараагийн алхмыг тодорхойлдог. Kafka нь лог хадгалах учраас бүх үйл явдал хадгалагдаж, үнэлгээ, дахин сургалтад ашиглаж болно. Агент бүрийн үйл явдал хадгалагдаж, олон шатлалт хянаж, шинжилгээ хийх боломжтой болно.

Энэ бол хиймэл оюун системийн шинэ фреймворк ба харин найдвартай, өргөжих чадвартай, бүрэн салангид байж чадах учраас шинэ стандарт болох боломжтой юм.

3.3 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. [22] [22] [24].

Хүснэгт 3.1: Ижил төстэй системүүдийн харьцуулалт

Шинж чанар	Inngest	Temporal	Camunda
Төрөл	Serverless workflow платформ	Durable execution платформ	BPMN процессын автоматжуулалт
Архитектур	EDA, хэрэглэгчийн функцийн түвшинд	Дэд процессийн түвшинтэй, олон хэлний дэмжлэг	BPMN 2.0 стандарт, enterprise
Хиймэл оюун дэмжлэг	LLM интеграци байдаг ч, RAG дутмаг байна	Тодорхой процессийг л гүйцэтгэх тул, өөрийгөө удирдах агент болч чадахгүй	Байхгүй
Байршуулалт	Serverless (вендороор)	Нарийн төвөгтэй (олон бүрэлдэхүүн)	Хүнд

Хүснэгт 3.1: Ижил төстэй системүүдийн харьцуулалт (үргэлжлэл)

Шинж чанар	Inngest	Temporal	Camunda
Урсгалын боловсруулалт	Хязгаарлалттай	Хязгаарлалттай	Хязгаарлалттай (BPMN-ын цар хүрээнд хязгаарлагдмал)
Ашиглагчид	Startup, жижиг баг	Uber, Netflix, Stripe	Томоохон байгууллагууд

3.3.1 Бидний санал болгож буй загварын ялгаа

Дээрх системүүдээс ялгаатай нь бидний загвар нь дараах онцлогуудтай.

Хүснэгт 3.2: Бидний санал болгож буй загварын онцлог

Онцлог	Тайлбар
Хиймэл оюун агентуудын интеграц	LLM, RAG, зааврын инженерчлэл, агентын төлөвлөгч зэргийг дотоод сиситемдээ интеграц найдвартай байдлаар интеграц хийж болдог.
Kafka-Flink суурилсан EDA	Өгөгдлийн бодит цагийн боловсруулалт, хэвтээ өргөжүүлэлтийн дэмжлэг, үйл явдлын хадгалалт хийснээр дахин ачааллуулж, үнэлэх боломжтой
Ухаалаг байгуулалт	Агент бүр хоорондоо өөр өөр зориулалттай мэдээллүүд солилцох ба маш нарийн даалгавруудыг гүйцэтгэж чадна
Нээлттэй эх	Kafka, Flink, PostgreSQL, Redis зэрэг бүрэн нээлттэй эх тул өөрийн бизнесийн онцлогоор хөгжүүлэлт хийх боломжтой
Мэдээллийн байдал	Суурь модел нь гаралтад суурилсан тул гарч болох эмзэг мэдээллийг хэрэглэгчдэд хүргэхээс өмнө хааж болно

Хүснэгт 3.2: Бидний санал болгож буй загварын онцлог (үргэлжлэл)

Онцлог	Тайлбар
Өргөтгөл	Шинэ агентууд нэмж, зохион байгуулагч ба RAG-д уг агентийн чадамжийг бүртгэж хэрэглээнд амар байдлаар нэвтэрнэ

3.4 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

4.1 Удиртгал

Энэхүү бүлэгт өмнөх бүлгүүдэд судалсан онолын зарчмуудыг практикт хэрэгжүүлсэн демо системийг дэлгэрэнгүй тайлбарлана. Хэрэгжүүлсэн систем нь Монголын Хөрөнгийн Бирж болон олон улсын хөрөнгийн зах зээлийн мэдээлэлд үндэслэн хиймэл оюун агентуудыг микросервис архитектурт event-driven байдлаар нэвтрүүлсэн санхүүгийн шинжилгээний веб платформ юм.

Систем нь хэрэглэгчдэд хувьцааны зах зээлийн талаарх мэдээллийг хялбар, ойлгомжтой байдлаар хүргэхийн зэрэгцээ хиймэл оюуны чадавхийг ашиглан персоналчилсан дүн шинжилгээ, зөвлөмж өгөх боломжийг олгодог. Уг хэрэгжүүлэлт нь зөвхөн онолын үнэн зөв байдлыг батлаад зогсохгүй bachelor диплом ажлын хүрээнд бодит ашиглагдах чадвартай технологийн шийдлийг харуулна.

4.2 Төслийн зорилго

Демо систем нь дараах гол зорилтуудыг хэрэгжүүлэхээр төлөвлөгдсөн.

4.2.1 Үндсэн зорилго

Үйл явдлаар удирдагдах агентын микросервис архитектурын давуу талыг бодит практикт харуулах. Онолын хэсэгт судалсан хиймэл оюун агент, RAG систем, event-driven архитектур, Apache Kafka болон Apache Flink зэрэг технологиудыг нэгтгэн уян хатан, өргөжих боломжтой, найдвартай систем бүтээх. Монголын Хөрөнгийн Биржийн бодит өгөгдөл ашиглан хэрэглэгчдэд практик үнэ цэнэтэй үйлчилгээ үзүүлэх боломжийг харуулах.

4.2.2 Техникийн зорилтууд

Эхний зорилт нь хиймэл оюун агентуудыг бие даасан микросервис болгон хөгжүүлэх явдал юм. Агент бүр салангид байж, өөрийн тодорхой үүрэгтэй, Docker контейнерт ажиллах ёстой. Хоёр дахь зорилт нь Apache Kafka ашиглан агентуудын хоорондын харилцааг event-driven байдлаар хэрэгжүүлэх. Энэ нь NxM нягт холбоосыг N+M болгон бууруулж, системийн уян хатан байдлыг нэмэгдүүлнэ. Гурав дахь зорилт нь RAG систем ашиглан хиймэл оюунд бодит, үнэн зөв мэдээллээр хангах. Дөрөв дэхь зорилт нь Next.js 14 ашиглан орчин үеийн, responsive веб интерфэйс бүтээх. Тав дахь зорилт нь МХБ-ийн бодит өгөгдөл ашиглан монгол хэрэглэгчдэд тохирсон функцүүдийг хэрэгжүүлэх явдал юм.

4.3 Системийн функционал

Хэрэгжүүлсэн систем нь дараах гол функцуудыг дэмждэг.

4.3.1 Хэрэглэгчийн удирдлага

Хэрэглэгчийн бүртгэл ба нэвтрэлтийн хувьд систем нь имэйл, нууц үг ашиглан бүртгүүлэх боломжийг олгодог. Бүртгүүлэх үед хэрэглэгч хөрөнгө оруулалтын зорилго (өсөлт эсвэл орлого), эрсдлийн хүлээх чадвар (бага, дунд, өндөр), сонирхож буй салбаруудыг (технологи, санхүү, уул уурхай гэх мэт) оруулна. Нууц үг нь bcrypt ашиглан hash хийгдэж, аюулгүй байдлыг хангана. Амжилттай бүртгүүлсний дараа JWT token үүсгэгдэж, 7 хоногийн хүчинтэй хугацаатай болно. Шинээр бүртгүүлсэн хэрэглэгчдэд Google Gemini AI ашиглан түүний профайлд тохирсон персоналчилсан өглөөний мэдээлэл автоматаар илгээгдэнэ. Энэ имэйл нь хэрэглэгчийн сонирхож буй салбар, эрсдлийн хүлээх чадварт нийцсэн зөвлөмж, анхаарах зүйлсийг агуулна.

Хэрэглэгчийн профайл удирдлагын хувьд хэрэглэгч өөрийн хөрөнгө оруулалтын зорилго, эрсдлийн хүлээх чадвар, сонирхож буй салбаруудаа хэдийд ч өөрчлөх боломжтой. Эдгээр өөрчлөлт нь дараагийн AI дүн шинжилгээ, зөвлөмжүүдэд шууд нөлөөлнө.

4.3.2 Watchlist cuctem

Watchlist удирдлагын хувьд хэрэглэгч олон watchlist үүсгэх боломжтой. Жишээлбэл "Миний уул уурхайн хувьцаа", "Технологийн компаниуд", "Урт хугацааны хөрөнгө оруулалт" гэх мэт нэршлээр watchlist үүсгэж болно. Watchlist бүрт Монголын болон олон улсын хувьцаануудыг нэмэх боломжтой. Монголын хувьцааны жишээ нь APU (Asia Pacific United), TDB (Trade and Development Bank), ERDENET зэрэг байж болно. Олон улсын хувьцааны жишээ нь AAPL (Apple), MSFT (Microsoft), TSLA (Tesla) зэрэг байна. Watchlist-ээс хувьцаа хасах, watchlist устгах зэрэг бүх үндсэн CRUD үйлдлүүд дэмжигдэнэ.

4.3.3 Хиймэл оюунтай харилцах

AI query интерфэйс нь системийн гол онцлог функц юм. Хэрэглэгч байгалийн хэл дээр асуулт асууж болно. Жишээ нь "APU хувьцааны сүүлийн үнэ, арилжааны хэмжээг харуул", "Уул уурхайн салбарын ерөнхий нөхцөл байдлыг тайлбарла", "Технологийн хувьцаанд одоо хөрөнгө оруулах нь зөв үү?" гэх мэт асуултууд асууж болно.

Orchestrator Agent нь хэрэглэгчийн асуултыг Google Gemini AI ашиглан ангилж, тохирох агент руу чиглүүлнэ. Хэрэв асуулт нь хувьцааны дүн шинжилгээтэй холбоотой бол Investment Agent руу, мэдээтэй холбоотой бол News Agent руу, ерөнхий мэдээлэлтэй холбоотой бол Knowledge Agent руу чиглүүлэгдэнэ. Агентууд нь МХБ-ийн бодит өгөгдөл (сүүлийн үнэ, арилжааны хэмжээ, өөрчлөлтийн хувь гэх мэт) ашиглан контекстэд нийцсэн, үнэн зөв хариулт үүсгэнэ. Хариулт нь Server-Sent Events ашиглан бодит цагт stream хэлбэрээр frontend руу дамждаг.

4.3.4 Мэдээний үйлчилгээ

News систем нь хоёр гол функцтэй. Нэгт, хэрэглэгчийн watchlist-д байгаа хувьцаануудтай холбоотой мэдээг Finnhub API-аас татаж авдаг. Finnhub нь олон улсын санхүүгийн мэдээний платформ бөгөөд компани, салбар, зах зээлийн талаарх шинэ мэдээллийг real-time байдлаар өгдөг. Хоёрт, өглөө бүр автоматаар хэрэглэгч бүрт

персоналчилсан мэдээний digest илгээгдэнэ. Энэ digest нь хэрэглэгчийн watchlist-тэй холбоотой хамгийн чухал 5-7 мэдээг Gemini AI ашиглан хураангуйлж, sentiment analysis (эерэг, сөрөг, төвийг сахисан) хийж, ойлгомжтой тайлбар нэмсэн байдаг. HTML email template ашигласан байдаг учир зураг, форматтай, уншихад тохиромжтой байдаг.

4.3.5 Мэдлэгийн сан

Knowledge Agent нь RAG (Retrieval-Augmented Generation) систем ашиглан хэрэглэгчдэд санхүүгийн мэргэжлийн мэдлэг өгдөг. МХБ-ийн компаниудын дэлгэрэнгүй мэдээлэл, арилжааны цаг, салбарын онцлог, хөрөнгө оруулалтын зарчим зэрэг мэдээлэл vector embedding хэлбэрээр хадгалагдсан байдаг. Sentence-Transformers модел ашиглан текстийг 384 хэмжээст vector болгон хөрвүүлдэг. PostgreSQL-ийн pgvector extension ашиглан cosine similarity хайлт хийж, асуулттай хамгийн холбоотой мэдээллийг олж авдаг. Энэ мэдлэг нь Investment Agent-ийн хариултыг баяжуулж, илүү нарийвчлалтай, үнэн зөв болгодог.

4.4 Хэрэглэгчийн шаардлага

Системийн хөгжүүлэлтэд дараах хэрэглэгчийн шаардлагууд анхаарагдсан.

4.4.1 Функциональ шаардлага

Хэрэглэгчид аюулгүй нэвтрэх систем хэрэгтэй. JWT token ашиглан session удирдлага, bcrypt ашиглан нууц үгийн аюулгүй байдал хангагдана. Хэрэглэгчид өөрийн watchlist-ийг хялбар удирдах боломж хэрэгтэй. Watchlist үүсгэх, хувьцаа нэмэх, хасах, watchlist устгах үйлдлүүд интуитив байх ёстой. Хэрэглэгчид хувьцааны талаар энгийн хэл дээр асуулт асуух боломж хэрэгтэй. Систем нь асуултыг ойлгож, бодит өгөгдөлд үндэслэн хариулах ёстой. Хэрэглэгчид өөрсдийн сонирхож буй хувьцааны талаарх мэдээг өглөө бүр имэйлээр хүлээн авах хэрэгтэй. Мэдээ нь watchlist-д тохирсон, хураангуйлагдсан, ойлгомжтой байх ёстой.

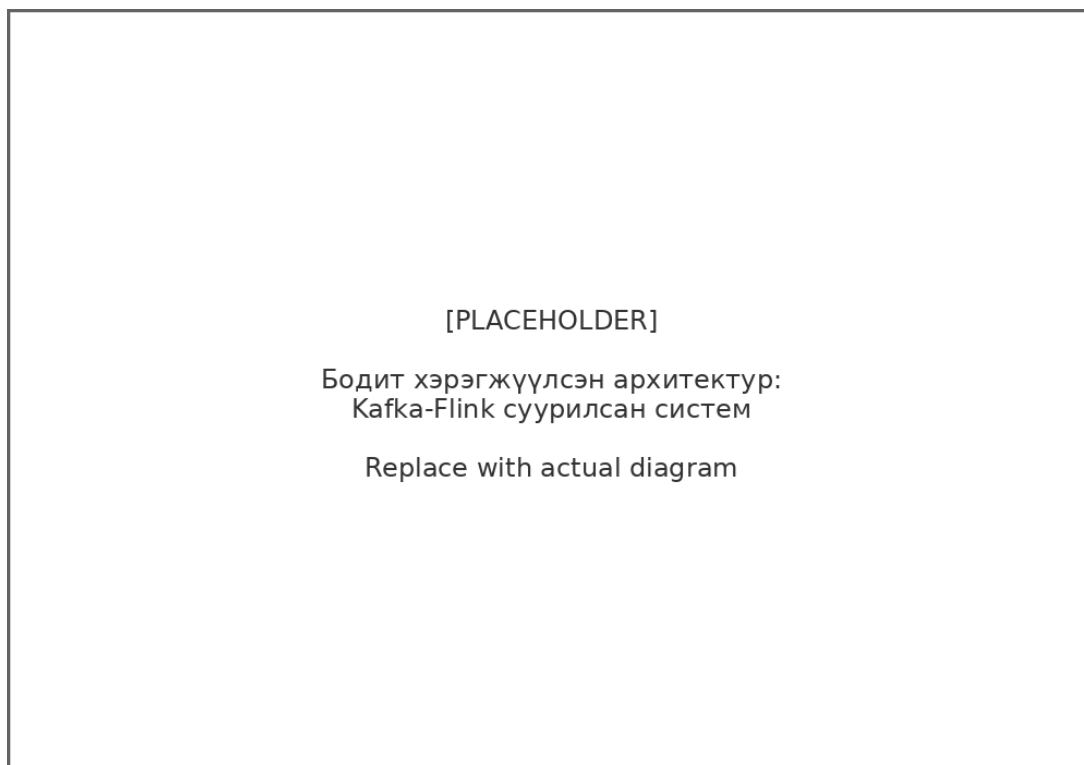
4.4.2 Техникийн шаардлага

Хариу өгөх хугацааны хувьд энгийн API хүсэлт 500ms-ээс бага, хиймэл оюуны боловсруулалт 15-20 секунд байх ёстой. Системийн найдвартай байдлын хувьд нэг агент унасан ч бусад агентууд хэвийн ажиллах ёстой. Kafka message persistence нь мэдээлэл алдагдахгүйг баталгаажуулна. Өргөжих чадварын хувьд систем нь Kafka partition mechanism ашиглан горизонтал өргөжих боломжтой байх ёстой. Агент нэмэх нь хялбар, систем дахин тохируулалт шаарддаггүй байх ёстой. Аюулгүй байдлын хувьд бүх нууц үг hash хийгдэх, JWT token HMAC-SHA256 ашиглан подпислогдох, бүх API endpoint CORS тохируулагдсан байх ёстой.

4.5 Системийн архитектур

Хэрэгжүүлсэн системийн архитектур нь event-driven microservices pattern дагаж зохион байгуулагдсан.

4.5.1 Ерөнхий архитектур



Зураг 4.1: Хэрэгжүүлсэн системийн архитектур

Хэрэглэгч Next.js 14 frontend-ээр системд хандана. Frontend нь API Gateway (Express.js, Node.js 20) руу HTTP/REST хүсэлт илгээнэ. API Gateway нь үүнийг хүлээн авч, Apache Kafka (3.5) руу event хэлбэрээр дамжуулна. Kafka нь төв мэдээллийн систем болж, бүх агентууд энэ дамжуулан харилцана. Orchestrator Agent нь user.requests topic-оос хүсэлт уншиж, Gemini AI ашиглан intent ангилж, тохирох агент руу чиглүүлнэ. Мэргэшсэн агентууд (Investment, Knowledge, News, PyFlink Planner) нь agent.tasks topic-оос даалгавраа уншиж, боловсруулж, agent.responses topic руу үр дүнгээ буцаана. API Gateway нь үр дүнг хүлээн авч, SSE (Server-Sent Events) эсвэл polling замаар frontend руу дамжуулна. PostgreSQL 16 нь users, watchlists, mse_companies, mse_trading_history, knowledge_base зэрэг өгөгдлийг хадгална. Redis 7 нь session болон response caching-д ашиглагдана.

4.5.2 Агентуудын үүрэг

Хүснэгт 4.1: Хэрэгжүүлсэн агентуудын үүрэг

Агент	Гол үүрэг	Технологийн чадвар
Orchestrator Agent	Зохион байгуулагч	Gemini AI-аар intent ангилах, агентуудыг идэвхжүүлэх, хариу нэгтгэх
Knowledge Agent	Мэдлэгийн санах ой	RAG систем, vector search, мэдээлэл баяжуулах
Investment Agent	Шинжилгээ	МХБ өгөгдөл дүн шинжилж, хөрөнгө оруулалтын зөвлөмж өгөх
News Agent	Мэдээний шүүлтүүр	Finnhub API, мэдээ цуглуулж хураангуйлах, sentiment analysis
PyFlink Planner	Төлөвлөгч	Үндсэн Kafka consumer/producer loop, энгийн routing

4.5.3 *Kafka Topics*

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

4.5.4 *Өгөгдлийн сангийн схем*

PostgreSQL database нь 10 гаруй хүснэгттэй. users хүснэгт нь хэрэглэгчийн мэдээлэл (email, password hash, investment profile) хадгална. watchlists хүснэгт нь хэрэглэгчийн үүсгэсэн watchlist-уудыг UUID primary key-тэй хадгална. watchlist_items хүснэгт нь watchlist дахь хувьцааны символуудыг хадгална. mse_companies хүснэгт нь МХБ-ийн

компаниудын мэдээлэл (symbol, нэр, салбар, индустри) хадгална. mse_trading_history хүснэгт нь өдөр бүрийн арилжааны түүх (нээлтийн үнэ, хаалтын үнэ, дээд/доод үнэ, хэмжээ) хадгална. knowledge_base хүснэгт нь RAG системд ашиглагдах мэдлэг, vector embedding-тэй хамт хадгалагдана. agent_responses_cache хүснэгт нь агентуудын хариултыг түр хадгалж, дахин ашиглах боломжийг олгоно. monitoring_events хүснэгт нь системийн лог, metric бичигдэнэ.

4.6 Технологийн стек

Системийн хөгжүүлэлтэд орчин үеийн, production-ready технологиуд ашиглагдсан.

4.6.1 Frontend Технологи

Хүснэгт 4.2: Frontend технологийн стек

Технологи	Хэрэглээ
Next.js 14	React-д суурилсан фреймворк, App Router, server-side rendering
React 18	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

4.6.2 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

Хүснэгт 4.3: Backend технологийн стек (үргэлжлэл)

Технологи	Хэрэглээ
Python 3.10	PyFlink Planner агентын хэл
Apache Kafka 3.5	Event-driven архитектурын үндэс, message broker
Zookeeper 3.8	Kafka кластерийн coordination
PostgreSQL 16	Үндсэн өгөгдлийн сан, pgvector extension-тэй
Redis 7	Session storage, response caching

4.6.3 Хиймэл оюуны технологи

Хүснэгт 4.4: Хиймэл оюуны технологи

Технологи	Хэрэглээ
Google Gemini 2.0 Flash	Intent classification, response generation, summarization
Sentence-Transformers	Text embedding (all-MiniLM-L6-v2 модел, 384-dim vectors)
pgvector	PostgreSQL-д vector хайлт, cosine similarity
Finnhub API	Олон улсын санхүүгийн мэдээний эх сурвалж

4.6.4 Deployment Технологи

Docker 24 болон Docker Compose 2.20 нь бүх сервисийг контейнержуулахад ашиглагдсан. Энэ нь хөгжүүлэлтийн орчин ба production орчны consistency хангадаг. Агент бүр өөрийн Docker контейнерт ажиллаж, бие даан өргөжих боломжтой. Ирээдүйд Kubernetes ашиглан production орчинд deploy хийх бэлтгэл хийгдсэн. GitHub Actions ашиглан CI/CD pipeline үүсгэх боломжтой.

4.7 Хэрэглээний тохиолдол

Системийн бодит ашиглалтыг дараах хэрэглээний тохиолдлуудаар харуулья.

4.7.1 *Шинэ хэрэглэгч бүртгүүлэх*

Хэрэглэгч registration хуудас руу орж, имэйл (demo@example.com), нууц үг, нэр (Болд) оруулна. Дараа нь хөрөнгө оруулалтын зорилго (Өсөлт), эрсдлийн хүлээх чадвар (Дунд), сонирхож буй салбар (Технологи, Санхүү) сонгоно. API Gateway нь POST /api/users/register endpoint-д хүсэлт хүлээн авна. Нууц үг нь bcrypt ашиглан hash хийгдэж, хэрэглэгч PostgreSQL-д хадгалагдана. JWT token үүсгэгдэж, 7 хоногийн хүчинтэй хугацаатай болно. Kafka руу user.events topic-т user.registered event publish хийгдэнэ. Email Service нь уг event-ийг хүлээн авч, хэрэглэгчийн профайл (Өсөлт зорилготой, дунд эрсдэл, технологи ба санхүүгийн салбар сонирхдог) ашиглан Gemini AI-аар персоналчилсан өглөөний мэдээлэл үүсгэнэ. Имэйлд "Технологийн салбарын хувьцаанд анхаарах нь таны өсөлтийн зорилгод нийцнэ", "Дунд эрсдэлийн профайлд тохирсон портфолио хэрхэн бүтээх вэ" зэрэг зөвлөмжүүд багтана. Имэйл амжилттай илгээгдсэний дараа хэрэглэгч token-тэй системд нэвтрэнэ.

4.7.2 *Watchlist үүсгэж хувьцаа нэмэх*

Хэрэглэгч dashboard руу нэвтэрч, "Миний уул уурхайн хувьцаа" нэртэй watchlist үүсгэнэ. POST /api/watchlist хүсэлт JWT token-тэй илгээгдэнэ. API Gateway нь хэрэглэгчийг authenticate хийж, watchlist-ийг UUID primary key-тэй PostgreSQL-д хадгална. Kafka руу watchlist.created event publish хийгдэнэ. Хэрэглэгч watchlist-даа APU (Монголын хувьцаа) болон BHP (олон улсын уул уурхайн хувьцаа) нэмэхээр шийднэ. POST /api/watchlist/:id/items хүсэлт хувьцаа бүрийн хувьд илгээгдэнэ. Watchlist items PostgreSQL-д хадгалагдаж, watchlist.item.added event publish хийгдэнэ. Хэрэглэгч watchlist-ээ нээж харахад APU болон BHP хоёр хувьцааг харна. Хожим нь BHP-г хасахаар шийдвэл DELETE /api/watchlist/:id/items/BHP хүсэлт илгээгдэж, watchlist.item.removed event publish хийгдэнэ.

4.7.3 Хувьцааны дүн шинжилгээ асуух

Хэрэглэгч AI chat interface руу орж "APU хувьцааны сүүлийн үнэ, арилжааны хэмжээг харуулаад ирээдүйд өсөх магадлал байгаа юу?" гэж асуулт асууна. Frontend нь POST /api/agent/query хүсэлт илгээнэ. API Gateway нь unique requestId үүсгэж, user.requests topic руу event publish хийнэ. Frontend нь шууд requestId-тэй хариу авч, SSE холболт үүсгэнэ. Orchestrator Agent нь user.requests topic-оос event consume хийнэ. Gemini AI ашиглан intent-ийг "market_analysis" гэж ангилна. Investment Agent руу agent.tasks topic-т даалгавар илгээнэ. Investment Agent нь agent.tasks topic-оос даалгавраа consume хийнэ. PostgreSQL-ээс APU хувьцааны мэдээллийг query хийнэ: "SELECT symbol, name, last_price, change_percent, volume FROM mse_companies WHERE symbol = 'APU'". Олдсон өгөгдөл (APU, 1280 MNT, +2.4%, 125340 ширхэг) ашиглан prompt үүсгэнэ. Gemini AI руу prompt илгээж, дүн шинжилгээтэй хариулт авна. Хариулт нь agent.responses topic руу publish хийгдэж, мөн agent_responses_cache хүснэгтэд хадгалагдана. API Gateway нь agent.responses topic-оос хариултыг consume хийж, SSE холболтоор frontend руу stream хийнэ. Frontend нь хариултыг chat interface-д харуулна. Хариулт нь "APU хувьцааны сүүлийн үнэ 1,280 MNT, өмнөх өдрөөс 2.4% өссөн байна. Арилжааны хэмжээ 125,340 ширхэг нь дундаж түвшинтэй. Уул уурхайн салбарын ерөнхий өсөлтийн чиг хандлагыг харгалзан үзвэл дунд хугацаанд өсөх боломжтой" гэх мэт агуулгатай байна.

4.7.4 Өглөөний мэдээний digest хүлээн авах

Өглөө 9:00 цагт cron job POST /api/daily-news/send endpoint дуудагдана. Систем нь PostgreSQL-ээс бүх идэвхтэй хэрэглэгчдийг query хийнэ. Хэрэглэгч бүрийн watchlist-ийн хувьцааны символуудыг авна. Хэрэглэгчийн watchlist-д APU, TDB, AAPL, MSFT байвал эдгээр хувьцаануудтай холбоотой мэдээг Finnhub API-аас татна. Сүүлийн 24 цагийн 20-30 нийтлэлийг цуглуулж, давхардсан мэдээг хасаж, хамгийн чухал 5-7 нийтлэлийг сонгоно. Gemini AI нь мэдээг хураангуйлж, bullet point хэлбэрээр гол сэдвүүдийг тодруулна. Sentiment analysis хийж (эерэг/сөрөг/төвийг сахисан) илэрхийлнэ. HTML email template ашиглан мэдээний digest бүтээнэ. Имэйлд зах зээлийн ерөнхий

байдал, хэрэглэгчийн watchlist-ийн хувьцаануудын гол мэдээ, анхааруулга, сонирхолтой дэлгэрэнгүй мэдээллийн холбоос зэрэг багтана. Email Service хэрэглэгчид имэйл илгээнэ. Хэрэглэгч өглөө сэрэхдээ өөрийн watchlist-тэй холбоотой хамгийн чухал мэдээллийг хүлээн авсан байна.

4.8 Хөгжүүлэлтийн явц

4.8.1 Хугацаа ба шат

Системийн хөгжүүлэлт 2024 оны 11 дүгээр сараас 2025 оны 1 дүгээр сар хүртэл 6 долоо хоногт явагдсан. Эхний 2 долоо хоногт infrastructure (Docker, Kafka, PostgreSQL, Redis) тохируулагдаж, database схем зохиогдож, Kafka topics үүсгэгдсэн. Дараагийн 2 долоо хоногт агентуудын үндсэн функцууд (Orchestrator-ийн intent classification, Knowledge-ийн RAG систем, Investment-ийн МХБ өгөгдөл интеграц, News-ийн Finnhub холболт) хөгжүүлэгдсэн. Дараа нь 1 долоо хоногт API Gateway-ийн бүх endpoints (users, watchlist, agent query, monitoring), JWT authentication, SSE streaming хэрэгжүүлэгдсэн. Эцсийн долоо хоногт Frontend Next.js 14 дээр хөгжүүлэгдэж (authentication UI, dashboard, chat interface, watchlist management), МХБ-ийн seed өгөгдөл нэмэгдэж, эцсийн тестүүд хийгдсэн.

4.8.2 Одоогийн байдал

Системийн хэрэгжилт ойролцоогоор 70% хийгдсэн байна. Бүрэн хэрэгжсэн хэсгүүд нь Infrastructure (Docker, Kafka, PostgreSQL, Redis бүх тохируулагдсан), Агентууд (Orchestrator, Knowledge, Investment, News, PyFlink Planner үндсэн функцүүдтэй), API Gateway (бүх endpoints ажилладаг), Frontend (authentication, dashboard, chat interface, watchlist management бэлэн), Database (МХБ-ийн seed өгөгдөлтэй) зэргийг агуулна. Хэсэгчлэн хэрэгжсэн хэсгүүд нь PyFlink Planner-ийн нарийн төвөгтэй функцууд (stateful computation, windowing), Frontend-ийн МХБ market overview (layout бэлэн, real-time updates дутмаг), Stock detail pages (үндсэн бүтэц, charts дутмаг), User settings (үндсэн функц, email preferences дутмаг), Monitoring (үндсэн agent status, Prometheus/Grafana дутмаг), Email (welcome ба daily news бэлэн, price alerts дутмаг) зэрэг байна. Хараахан эхлээгүй хэсгүүд нь Portfo-

lio management, Risk assessment (VaR тооцоолол), Advanced analytics (correlation analysis), Real-time market data (WebSocket), Machine learning features (price prediction), Production deployment (Kubernetes, load balancer, auto-scaling) зэрэг ирээдүйн хөгжүүлэлтэд үлдсэн.

4.8.3 Туршилтын үр дүн

Системийн туршилт нь бүх үндсэн функцүүд ажиллаж байгааг баталгаажуулсан. Хэрэглэгчийн бүртгэл нь welcome email-тэй амжилттай. Хэрэглэгчийн нэвтрэлт нь JWT token-тэй ажилладаг. Watchlist CRUD үйлдлүүд бүгд ажилладаг. AI agent query submission болон response хүлээн авалт амжилттай. Event-driven урсгал (Kafka → Orchestrator → Investment Agent → Response) бүрэн ажилладаг. SSE streaming болон polling хоёулаа ажилладаг. Monitoring API нь 5/5 агентыг active гэж зөв харуулж байна. Daily news email илгээлт амжилттай.

Гүйцэтгэлийн хувьд database queries 50-100ms, Kafka message delivery 5-10ms, API Gateway endpoints 200-500ms, LLM inference 10-20 секунд, нийт end-to-end урсгал (AI-тай) ойролцоогоор 17 секунд байна. Эдгээр үр дүн нь bachelor диплом ажлын демод хангалттай гэж үзэж болно.

4.9 Бүлгийн дүгнэлт

Энэхүү бүлэгт санал болгосон event-driven агентын микросервис загварыг бодитоор хэрэгжүүлсэн системийн дэлгэрэнгүй тайлбарыг өгсөн. Хэрэгжүүлэлт нь Монголын Хөрөнгийн Биржийн өгөгдөлд суурилсан санхүүгийн шинжилгээний вэб аппликейшн болж, онол практикийн холбоог тодорхой харуулсан.

Системийн үндсэн зорилго нь event-driven агентын микросервис архитектурын давуу талыг бодит практикт харуулах явдал байсан. Энэ зорилго нь 70% хэрэгжилттэй амжилттай биелсэн гэж үзэж болно. Хэрэглэгчийн удирдлага, watchlist систем, хиймэл оюунтай харилцах интерфэйс, мэдээний үйлчилгээ, мэдлэгийн сан зэрэг гол функцууд бүрэн ажиллаж байна.

Технологийн стек нь орчин үеийн, production-ready технологиудаас бүрдсэн. Frontend талаас Next.js 14, React 18, TypeScript 5, Tailwind CSS ашиглагдсан. Backend талаас Node.js 20, Express.js, Python 3.10 ашиглагдсан. Infrastructure болгон Apache Kafka 3.5, PostgreSQL 16, Redis 7 сонгогдсон. Хиймэл оюуны технологи болгон Google Gemini 2.0 Flash, Sentence-Transformers ашиглагдсан. Бүх эдгээр технологиуд нь Docker контейнерт ажиллаж, хөгжүүлэлтийн орчин ба production орчны consistency хангадаг.

Системийн архитектур нь онолын бүлгүүдэд судалсан зарчмуудыг шууд дагаж зохион байгуулагдсан. Event-driven architecture нь агентуудын хоорондын NxM нягт холбоосыг N+M болгон бууруулж, салангид байдлыг хангасан. Orchestrator Agent нь төв зохион байгуулагч болж, мэргэшсэн агентууд нь өөр өөрийн үүргээ гүйцэтгэж байна. RAG систем нь Knowledge Agent-д мэргэжлийн мэдлэг өгч, Investment Agent-ийн хариултыг баяжуулж байна. Apache Kafka нь бүх харилцааны үндэс суурь болж, найдвартай, өргөжих боломжтой систем бүтээхэд чухал үүрэг гүйцэтгэж байна.

Хэрэглээний тохиолдлууд нь системийн бодит хэрэглээг харуулсан. Хэрэглэгч бүртгүүлэхээс эхлээд AI дүн шинжилгээ авах хүртэлх бүх урсгал асинхрон, event-driven байдлаар ажиллаж байгаа нь харагдаж байна. Энэ нь зөвхөн онолын хувьд биш, практикт ч event-driven архитектур үр дүнтэй ажилладгийг баталгаажуулж байна.

Хөгжүүлэлтийн явц нь 6 долоо хоногт явагдаж, 70% хэрэгжилтэд хүрсэн нь bachelor диплом ажлын хүрээнд хангалттай ахиц дэвшил юм. Үлдсэн 30% нь portfolio management, risk assessment, machine learning features зэрэг нарийн төвөгтэй функцууд бөгөөд эдгээр нь ирээдүйн хөгжүүлэлт эсвэл магистрын түвшний судалгаанд тохиромжтой.

Туршилтын үр дүн нь бүх үндсэн функцууд ажиллаж, event-driven урсгал бүрэн хэрэгжиж, гүйцэтгэл хүлээн зөвшөөрөгдөх түвшинд байгааг харуулсан. Энэ нь санал болгосон загварын үр ашигтай байдлыг практикаар баталгаажуулсан юм.

Эцэст нь энэхүү хэрэгжүүлэлт нь хиймэл оюуны инженерчлэл, агентын архитектур, микросервис, event-driven architecture зэрэг онолын зарчмууд бодит практикт амжилттай хэрэглэгдэж болохыг тодорхой харуулсан. Систем нь зөвхөн bachelor диплом ажлын demo биш, харин цаашид хөгжүүлж, production орчинд нэвтрүүлэх боломжтой суурь болсон гэж дүгнэж болно.

Дүгнэлт

Энэхүү судалгааны ажил нь хиймэл оюун агентуудыг микросервис архитектурт нэвтрүүлэх боломжийг судалж, практик загвар санал болгосон. Судалгааны явцад дараах гол үр дүнд хүрсэн:

Онолын хувьд:

Хиймэл оюуны инженерчлэл нь програм хангамж хөгжүүлэлтийн шинэ салбар болж хөгжиж байгаа нь тодорхой. Суурь моделийн гарч ирэх нь аппликейшн хөгжүүлэлтийн саад бэрхшээлийг эрс багасгасан. Хэл загваруудаас том хэлний загвар, цаашлаад суурь загвар руу хөгжих үйл явц нь арван жилийн технологийн дэвшлийн үр дүн юм. Өөрийгөө удирдсан сургалт, Transformer архитектур, post-training 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 үйлдвэрлэлийн түвшинд өргөжүүлэхийн тулд зөвхөн агентын ур чадвар биш, тэдгээрийг холбодог архитектур чухал гэдгийг ойлгосон. Монолит болон RPC/API суурилсан холболт нь монолит архитектурт тулгарсан асуудалтай адил

саад болдог. Үүнийг шийдэхийн тулд үйл явдлаар Event-Driven архитектур ашиглан агентуудыг салангид микросервис болгон хөгжүүлэх шаардлагатай.

Event-Driven архитектур

Apache Kafka болон Apache Flink ашиглан event-driven агентын систем бүтээх нь:

- **Салангид байдал:** Kafka topics-оор харилцах нь агентууд хоорондоо шууд хамааралгүй байхыг баталгаажуулна
- **Параллель боловсруулалт:** Kafka partitions ашиглан олон агент зэрэг ажиллаж, системийн дамжуулалтыг нэмэгдүүлнэ
- **Найдвартай байдал:** Үйл явдлын хадгалалт нь мэдээлэл алдагдахгүй, дахин тоглуулж болохыг баталгаажуулна
- **Водит цагийн боловсруулалт:** Streaming архитектур нь тэр даруй хариу үйлдэл үзүүлэх боломжийг олгоно
- **Олон хэрэглэгч:** Агентын гаралт нь CRM, CDP, analytics зэрэг олон системд автоматаар урсах боломжтой

Flink AI Inference ашиглах нь orchestrator агентыг stream processing app болгон хөгжүүлэх боломжийг олгож, Flink-ийн stateful боловсруулалт, exactly-once семантик зэрэг давуу талыг ашиглах боломжтой болгоно.

Apache Kafka болон Apache Flink зэрэг технологиудыг ашиглан event-driven агентын систем нь:

- Бие даан өргөжиж чадна
- Бодит цагт өгөгдөл боловсруулна
- Системийн хэмжээнд мэдээллээ хуваалцана
- Алдаанаас сэргэж чадна
- Дахин тоглуулах замаар сайжирна

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.

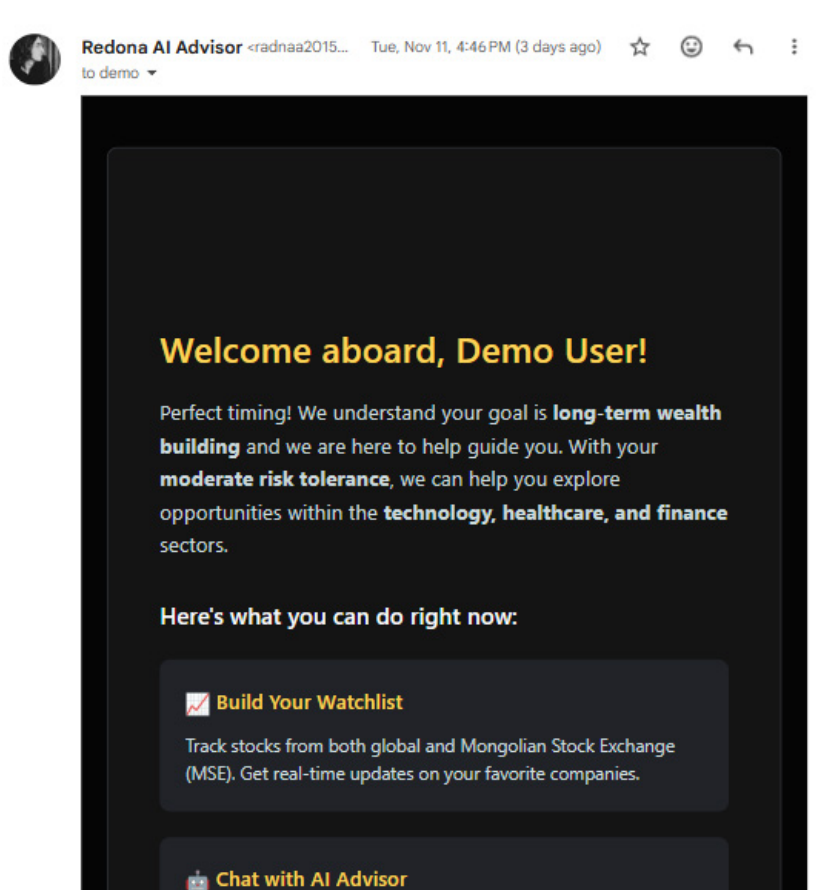
Энэхүү дипломын ажил нь хиймэл оюуны инженерчлэлийн үндсийг тавьж, микросервис архитектурт агент суурилсан шийдлийг практикт хэрхэн хэрэглэж болохыг харуулсан.

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>

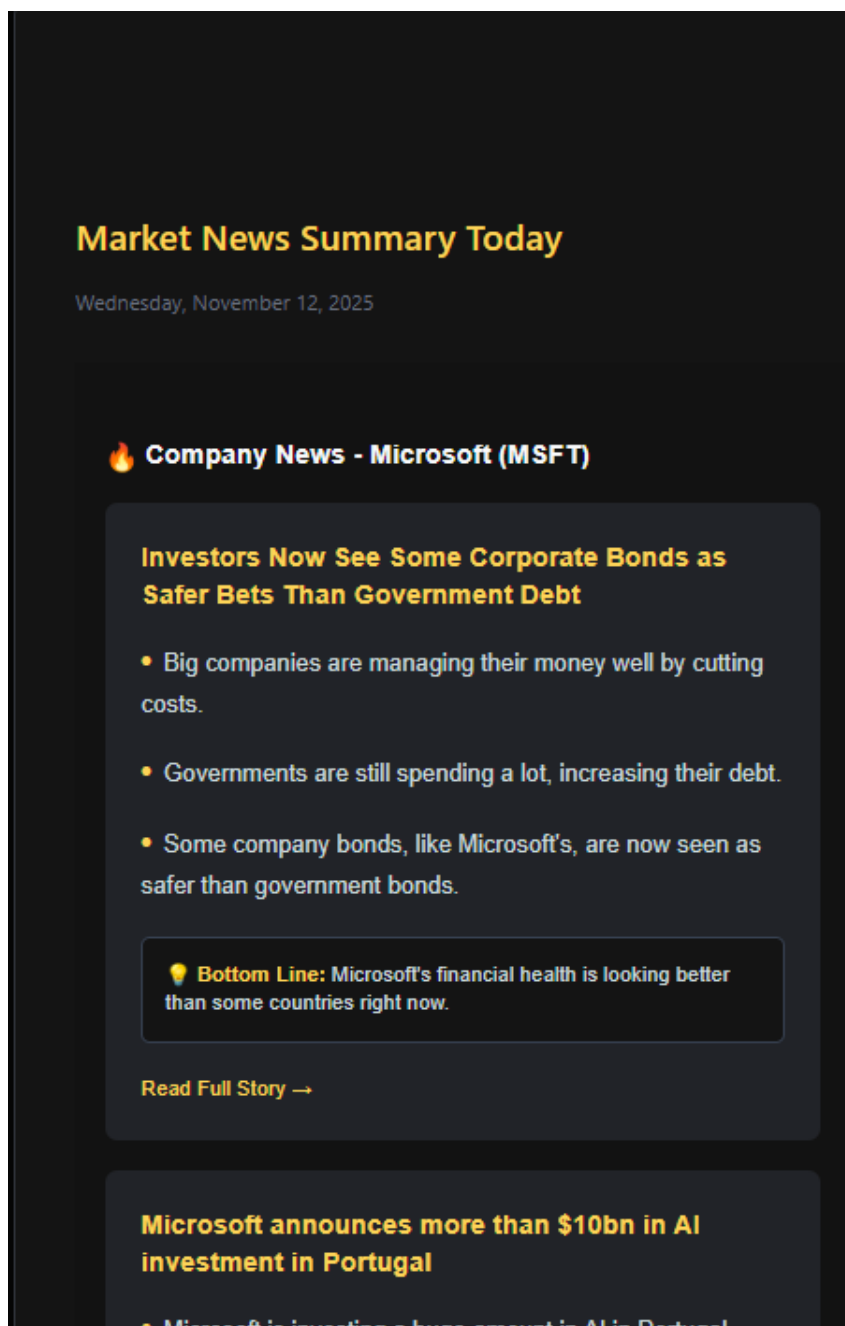
А. ӨӨРИЙН МЭДЭЭЛЭЛД ТОХИРСОН БАЙДЛААР БҮРТГҮҮЛЭЛТИЙН Э- МЭЙЛ АВАХ



Зураг А.1: RAG-ийн бүтэц

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

В. ӨӨРИЙН МЭДЭЭЛЭЛД ТОХИРСОН БАЙДЛААР ӨДӨР ТУТМЫН Э-МЭЙЛ АВАХ



Зураг В.1: RAG-ийн бүтэц

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