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

Микросервис архитектурт суурилсан хиймэл оюун агентууд

(AI agents for microservices)

Мэдээллийн технологи (D061304)
Дипломын ажлын тайлан

Удирдагч: _____ Дэд профессор Б.Сувдаа

Хамтран удирдагч: _____

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

ЗУРГИЙН ЖАГСААЛТ

ХҮСНЭГТИЙН ЖАГСААЛТ

Кодын жагсаалт

УДИРТГАЛ

Сүүлийн жилүүдэд хиймэл оюун ухааны салбар дахь технологийн хурдацтай хөгжил нь програм хангамж хөгжүүлэлтийн арга барилд үндсэндээ өөрчлөлт авчирсан. Тухайлбал, суурь моделийн (foundation models) гарч ирэх нь аппликейшн хөгжүүлэлтийн өмнө тулгардаг саад бэрхшээлийг эрс багасгасан бөгөөд энэ нь хиймэл оюуны инженерчлэл (AI engineering) 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 (machine learning) загвар бэлтгэхэд өндөр мэргэжлийн ур чадвар болон асар их өгөгдлийн иж бүрдэл шаардлагатай байсан бол одоо та бэлэн моделийг ашиглан аппликейшн хөгжүүлж чадна.

Энэхүү хөгжил нь микросервис архитектур дээр тулгуурласан програм хангамжийн хөгжүүлэлтэд онцгой боломжуудыг нээж өгч байна. Микросервис архитектур нь том нэг 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 (Large Language Models) 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 болон үйлдэл хийх механизм, мэдлэг нэмэгдүүлэх арга (Retrieval-Augmented Generation), болон эдгээрийг микросервис архитектурт хэрхэн нэгтгэх талаар авч үзэх юм.

0.1 Судалгааны зорилго

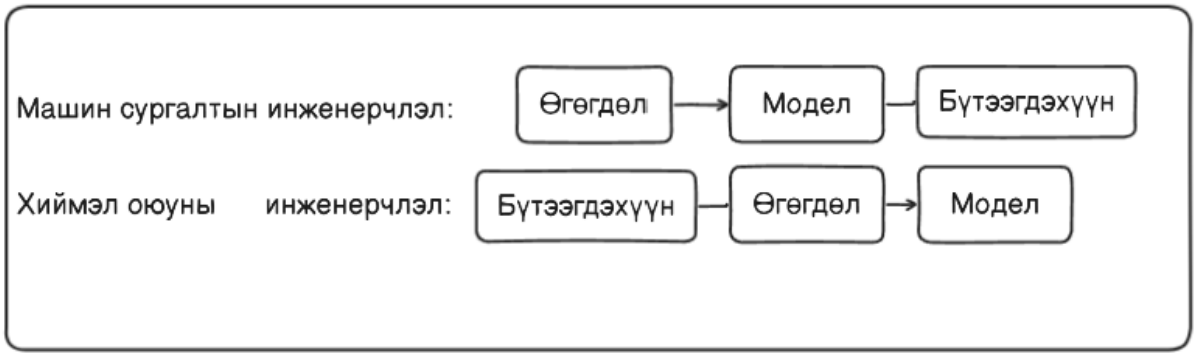
Энэхүү судалгааны ажлын гол зорилго нь дараах асуултуудад хариулна:

1. Хиймэл оюуны инженерчлэлийн үндсэн онолыг тайлбарлах
2. Хиймэл оюун агентуудын архитектур, төлөвлөлтийн механизм, хэрэглүүрүүдын ашиглалтыг судлах
3. Микросервис архитектурын давуу болон сул талуудыг тодорхойлох
4. Хиймэл оюун агентуудыг микросервис архитектурт нэвтрүүлэхэд тулгарах асуудлуудыг тодорхойлох
5. Хиймэл оюун агентуудыг микросервис архитектурт нэвтрүүлэх загвар санал болгох

1. ОНОЛЫН ХЭСЭГ

1.0.1 Хиймэл оюуны инженерчлэл гэж юу вэ

Хиймэл оюуны инженерчлэл гэдэг нь өмнө нь бэлтгэгдсэн суурь модел дээр аппликейшн хөгжүүлэх үйл явцийг хэлнэ. Энэ нь уламжлалт машин сургалтын инженерчлэл (ML engineering) эсвэл MLOps-оос ялгаатай байдаг. Хэрэв уламжлалт MS инженерчлэл нь загвар хөгжүүлэхэд чиглэсэн бол, хиймэл оюун инженерчлэл нь одоо байгаа моделийг ашиглахад илүү анхаарал хандуулдаг [1]. Хүчирхэг суурь



Зураг 1.1: Хиймэл оюуны инженерчлэл ба машин сургалтын инженерчлэл

моделийн олдоц болон хүртээмж нь дараах гурван хүчин зүйлийг бүрдүүлж, Хиймэл оюун инженерчлэл хурдан өсч буй салбар болоход хүргэсэн:

1. **Өндөр эрэлт**: 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 орлогын тайланд хиймэл оюуныг дурдсан тоо нь өмнөх оноос гурав дахин их байна.

2. **Хөгжүүлэлт**: 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.

3. **Том боломж**: Хиймэл оюун технологи нь ажил хэргийг автоматжуулалах, шинэ бүтээгдэхүүнүүдийг бий болгох зэрэг асар том боломжуудыг санал болгож байна.

Хиймэл оюунт аппликейшний нээлттэй эхийн кодууд (AutoGPT, Stable Diffusion WebUI, LangChain, Ollama) нь GitHub дээр Bitcoin-оос ч илүү од цуглуулжээ.

1.0.2 Суурь моделийн хөгжил

Хэл загваруудаас том хэлний загвар, суурь загвар руу хөгжих үйл явц нь хэдэн арван жилийн технологийн дэвшлийн үр дүн юм. Энэхүү хэсэгт гол түлхүүр цэгүүдийг авч үзэх болно.

Хэл моделийн үндэс

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

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

Хэл моделийн хоёр үндсэн төрөл байдаг:

- **Далдлагдсан хэл загварууд**: Өгүүлбэр доторх далдлагдсан үгсийг таамаглах замаар сурдаг. BERT нь энэ төрлийн алдартай жишээ юм.
- **Авторегрессив хэл загварууд**: Өмнөх токенуудад үндэслэн дараагийн токенийг таамаглах замаар сурдаг. Одоогийн ChatGPT, Cluade нь үүний жишээ юм.

Өөрийгөө удирдсан сургалт

Хэл моделийн хамгийн чухал давуу тал нь өөрийгөө удирдсан сургалт (self-supervision) ашиглах чадвар юм. Өөрийгөө удирдсан сургалт нь удирдлагатай сургалт (supervised

learning)-аас ялгаатай байдаг. Удирдлагатай сургалт нь тэмдэглэгдсэн өгөгдөл (labeled data) шаарддаг бөгөөд энэ нь үнэтэй, цаг хугацаа их зарцуулдаг.

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 Загвар (LLM) болтол өргөжүүлэх боломжтой болгосон.

Том хэлний загвараас суурь загвар руу

2017 онд Transformer архитектур гарч ирснээр хэл моделийн чадамж харьцангуй өндөр нэмэгдсэн. Attention механизм нь загваруудад урт хугацааны хамаарлыг илүү сайн ойлгох боломжийг олгосон.

Том хэлний загварууд (Large Language Models, LLMs) 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 загваруд (foundation models) нь LLM-ээс цааш өргөжсөн ойлголт юм. Эдгээр нь зөвхөн текст биш, зураг, аудио, видео зэрэг олон төрлийн өгөгдөл боловсруулж чаддаг том мульти модаль загварууд юм. Суурь моделийн гол онцлог нь тодорхой үүрэгтэй моделээс цаашлаад ерөнхий зориулалтын загвар руу шилжсэн нь юм.

1.0.3 Суурь моделийн сургалт

Суурь моделийг бэлтгэх нь хоёр үндсэн үе шаттай:

Урьдчилан сургалт

Урьдчилан сургалт нь өөрийгөө удирдсан сургалт ашиглан их хэмжээний өгөгдөл дээр моделийг сургах үйл явц юм. Энэ үе шатанд загвар нь хэл, ерөнхий мэдлэг, дүрэм, баримт бичгүүдээс суралцдаг. Гэвч энэхүү үе шатд хэрэглэгчдийн хүсэлтэд нийцсэн хариулт өгөхөд сайн биш байдаг. Учир нь харилцан яриа өрнүүлэх гэхээс илүүтэйгээр зөвхөн өгүүлбэрийн гүйцээлт рүү тулгуурлан сургагдсан байдаг.

Дараах сургалт

Урьдчилан сургасан моделийг хэрэглэгчдийн хүсэлтэд тохируулахын тулд дараах сургалт хийдэг. Энэ нь хоёр үе шаттай:

1. **Supervised Finetuning, SFT**: Өндөр чанартай зааварчилгааны өгөгдөл (instruction data) дээр моделийг нарийвчлан сургаж, гүйцээлт биш харин ярианы горимд оновчтой болгоно.

2. **Preference Finetuning**: Моделийг хүний сонголттой нийцсэн хариулт өгөхийн тулд цаашид нарийвчлан сургана. Үүнд RLHF (Reinforcement Learning from Human Feedback), DPO (Direct Preference Optimization), RLAIF (Reinforcement Learning from AI Feedback) зэрэг аргууд ордог.

1.0.4 Sampling стратегиуд

Модел нь гаралтаа sampling процессоор бүтээдэг. Sampling нь хиймэл оюуны гаралтын магадлалыг шууд нөлөөлдөг. Дараах параметрууд нь sampling-д нөлөөлдөг:

- **Temperature**: Температур өндөр байх тусам загвар илүү бүтээлч, гэнэтийн хариулт өгдөг. Температур бага байх тусам илүү таамагладаг, баттай хариулт өгнө.

- **Top-k**: Хамгийн их магадлалтай k ширхэг токеноос сонгодог. Үүнийг өөрчилснөөр хариулт содон байж болно.

- **Top-p (nucleus sampling)**: Нийлбэр магадлал нь p-д хүрэх хамгийн бага токенуудын багцаас сонгодог. Энэ нь тийм, үгүй, урт хариулт, богино хариултын загварыг тодорхойлж болдог. Гэхдээ токений хэрэглээ багасна гэсэн үг биш.

1.0.5 Моделийн үр дүнг хэмжих

Суурь моделийг үнэлэх нь эрсдэлийг бууруулах, цаашлаад боломжуудыг илрүүлэх тал дээр чухал ач холбогдолтой. Үнэлгээ нь загвар сонгох, дэвшлийг хэмжих, аппликейшн ашиглалтад бэлэн эсэхийг тодорхойлох, асуудал болон боломжуудыг илрүүлэх зэрэгт шаардлагатай.

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

Үнэлгээний гол асуудлууд:

- **Нээлттэй төгсгөлтэй гаралт**: Суурь загваруудыг зөвхөн гаралтуудын өгөгдлөөс дүгнэх нь тул үнэлэхэд хэцүү.
- **Тогтворгүй байдал**: Загвар нь ижил эсвэл бага зэрэг өөр асуулт асуухад маш өөр хариулт өгч болно. Үүнээс хиймэл оюуны суурь моделын хариулт нь магадлалаас үүсдгийг ажиглаж болно.
- **Төөрөгдөл**: Загвар нь баримт дээр үндэслээгүй буруу хариулт өгч болно.

1.1 Prompt инженерчлэл

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

1.1.1 Prompt бичих шилдэг арга барил

OpenAI-ийн санал болгосон дараах стратегиудыг дагах нь илүү сайн үр дүн өгдөг:

1. **Тодорхой зааварчилгаа өгөх**: Юу хийлгэхээ хоёрдмол утгагүй байдлаар тайлбарлах хэрэгтэй.
2. **Persona өгөх**: Загвараас тодорхой дүрд тоглуулж болно. Жишээ нь: ”Та туршлагатай программист. Кодыг шалгаад алдаа зааж өг.”
3. **Жишээ өгөх**: Жишээ өгснөөр хариултын формат болон стилийн хоёрдмол утгыг багасгадаг. Энэ нь few-shot learning гэгддэг.

4. **Хангалттай контекст өгөх**: Контекст нь hallucination-ийг багасгадаг. Хэрэв модел шаардлагатай мэдээллээр хангагдаагүй бол өөрийн дотоод мэдлэгтээ найдах бөгөөд энэ нь найдваргүй байж болно.

5. **Нарийн төвөгтэй даалгавруудыг хялбар дэд даалгавруудад хувааж өгөх**: Үр дүнтэйгээр бага токен ашиглана. chain-of-thought prompting гэх техник байдаг.

1.1.2 Хамгаалалтын prompt инженерчлэл

Аппликейшн олон нийтэд ашиглагдах болмогц гурван төрлийн довтолгооноос хамгаалах шаардлагатай:

- **Jailbreaking болон prompt injection**: Моделийн зөвшөөрөөгүй үйлдэл хийлгэх оролдлого.
- **Мэдээлэл задруулах**: Моделийн сургалтын өгөгдөл эсвэл контекстын мэдээллийг задруулах оролдлого.

1.2 Retrieval-Augmented Generation (RAG)

RAG буюу Retrieval-Augmented Generation нь моделийн мэдлэгийг гадаад эх сурвалжаар өргөтгөх арга юм. Энэ нь моделийн дотоод мэдлэг нь хангалтгүй, хуучирсан эсвэл алдаатай байх асуудлыг шийддэг.

Хэдийгээр моделийн контекстийн урт тогтмол нэмэгдэж байгаа ч RAG-ийн ач холбогдол алдагдахгүй байна:

1. Зарим аппликейшнд өгөгдлийн хэмжээ байнга өсч байдаг. Иймээс RAG-ийг сайтар үнэлж, тасралтгүй хөгжүүлэлт хийснээр бодит хэрэглээнд үр нөлөө алдахгүй байна.

2. Урт контекстыг боловсруулж чаддаг гэдэг нь тэр контекстыг сайн ашигладаг гэсэн үг биш. Контекст урт байх тусам загвар буруу хэсэгт анхаарал хандуулах магадлал өсдөг.

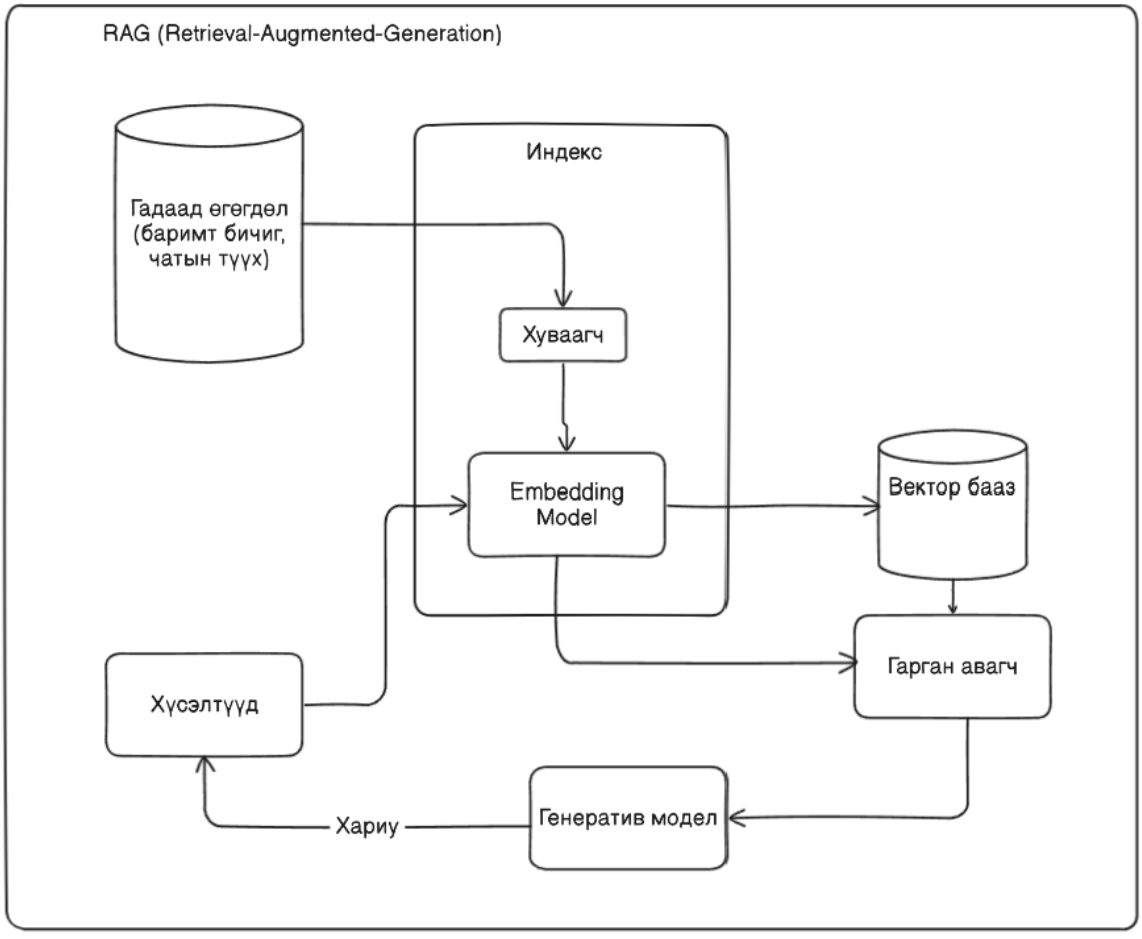
3. Контекстын токен бүр нэмэлт өртөг, нэмэлт latency авчирдаг. RAG нь асуулт бүрт зөвхөн хамгийн холбогдолтой мэдээллийг ашиглах боломжийг олгоно.

Anthropic-ийн зөвлөмжөөр хэрэв мэдлэгийн сан 200,000 токеноос бага (ойролцоогоор 500 хуудас бүхий өгөгдөл) бол RAG ашиглалгүй бүх мэдлэгийг prompt-д оруулж болно гэжээ.

1.2.1 RAG системийн бүтэц

RAG систем нь хоёр гол бүрэлдэхүүнтэй:

1. **Retriever (Хайгч)**: Асуултад хамгийн холбогдолтой баримтуудыг олж авдаг.
2. **Generator (Үүсгэгч)**: Олж авсан баримтуудыг асуултад ашиглан хариулт үүсгэдэг.



Зураг 1.2: RAG-ийн бүтэц

1.2.2 Retrieval алгоритмууд

Хайлтаар олдсон өгөгдөл нь хэр оновчтой байх нь RAG-ийн хамгийн чухал хэсгүүдийн нэг. 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 (Term-based retrieval)

Энэ арга нь түлхүүр үгээр баримт хайдаг. Энэ арга Google зэрэг хөтчийн хайлтын алгоритмд ашиглагдсаар ирсэн. Аргууд:

- **TF (Term Frequency)**: Нэр томьёо баримт доор хэдэн удаа гарч байгааг хэмждэг.
- **IDF (Inverse Document Frequency)**: Нэр томьёо хэдэн баримтад гарч байгааг үндэслэн түүний чухлыг хэмждэг.

Түгээмэл шийдлүүд: Elasticsearch, BM25. Эдгээр нь inverted index ашигладаг.

Embedding суурилсан хайлт (Embedding-based retrieval)

Semantic хайлт гэж нэрлэгддэг энэ арга нь утгын түвшинд холбоотой байдлаар тооцдог. Баримт бүр хуваагдаж vector embedding болгон хувиргагдаж, vector database-д хадгалагдана. Асуулт ирэх үед түүний embedding-тэй хамгийн ойр векторуудыг хайдаг.

Vector search алгоритмууд:

- **k-NN (k-Nearest Neighbors)**: Энгийн арга боловч өгөгдөл их бол удаан.
- **ANN (Approximate Nearest Neighbors)**: Хурдан боловч ойролцоогоор хайна.
- **LSH (Locality-Sensitive Hashing)**: Ижил төстэй векторуудыг нэг bucket-д hash хийнэ.
- **HNSW (Hierarchical Navigable Small World)**: Олон давхаргат граф ашиглана.

- **IVF (Inverted File Index)**: K-means clustering ашиглан векторуудыг бүлэглэнэ.

Алдартай vector database-үүд: FAISS, Milvus, Pinecone, Weaviate, Qdrant.

1.2.3 RAG-ын үнэлгээ

RAG системийг үнэлэхэд дараах метрикүүд ашигладаг:

- **Context Precision**: Олж авсан баримтуудын хэдэн хувь нь асуулттай холбоотой вэ?
- **Context Recall**: Асуулттай холбоотой бүх баримтуудын хэдэн хувийг олж авсан бэ?
- **Answer Quality**: Эцсийн хариултын чанар хэр сайн вэ?

1.2.4 RAG-ыг сайжруулах аргууд

1. **Chunking стратеги**: Баримтуудыг хэрхэн хэсэглэх нь чухал. Тогтмол урттай хэсэглэх, өгүүлбэр/догол мөрөөр хэсэглэх, semantic хэсэглэх зэрэг аргууд байдаг.
2. **Reranking**: Анхны хайлтын үр дүнг дахин эрэмбэлэн илүү нарийвчлалтай болгох.
3. **Query rewriting**: Асуултыг дахин найруулж илүү сайн хайлт хийх.
4. **Hybrid хайлт**: Нэр томьёо болон embedding суурилсан хайлтыг хослуулах.
5. **Contextual retrieval**: Хэсэг бүрийг metadata, түлхүүр үг, холбогдох асуултуудаар баяжуулах.

1.2.5 Агентын RAG (Agentic RAG)

Уламжлалт RAG нь тогтмол workflow ашигладаг. Асуулт ирэх бүрд ижил үйл явц дагана: хайлт хийх, контекст үүсгэх, хариулт үүсгэх. Гэвч энэ арга нь нарийн төвөгтэй даалгавруудад хязгаарлагдмал байдаг.

Агентын RAG нь RAG-ыг илүү динамик, контекст дээр суурилсан болгож хөгжүүлдэг. Тогтмол workflow-д найдахын оронд, агентууд нь бодит цагт ямар өгөгдөл хэрэгтэйгээ, хаанаас олохоо, өгөгдсөн даалгаврын үндсэн дээр асуултаа хэрхэн боловсруулах вэ гэдгийг шийддэг.

Агентын RAG vs Уламжлалт RAG:

- **Динамик хайлт**: Агент нь шаардлагатай бол олон эх сурвалжаас мэдээлэл цуглуулж, асуултаа боловсронгуй болгож, шинэ мэдээлэл гарч ирэхэд дасан зохицож чаддаг.
- **Олон алхам бүхий дүгнэлт**: Агент нь анхны хайлтын үр дүнд үндэслэн дараагийн асуултуудыг үүсгэж, илүү гүнзгий мэдлэг олж авч чаддаг.
- **Хэрэглүүрийн ашиглалт**: Агент нь зөвхөн баримтын хайлт биш, API дуудах, өгөгдлийн сангаас асуулт хийх, тооцоолол хийх зэрэг олон багаж ашиглаж болно.
- **Контекст санах**: Санах ойгоороо агент нь өмнөх асуултууд болон хариултуудыг хадгалж, дараагийн асуултдаа илүү нарийвчлалтай хандаж чаддаг.

Жишээлбэл, маркетингийн стратеги боловсруулж байгаа агент нь:

1. CRM-ээс харилцагчийн өгөгдөл татаж авна
2. API ашиглан зах зээлийн чиг хандлагыг цуглуулна
3. Шинэ мэдээлэл гарч ирэхэд өөрийн аргачлалаа боловсронгуй болгоно
4. Санах ой болон итераци ашиглан илүү нарийвчлалтай, холбогдолтой үр дүн гаргана

Агентын RAG нь хайлт, дүгнэлт, үйлдлийг нэгтгэдэг.

1.3 Хиймэл оюуны агентууд

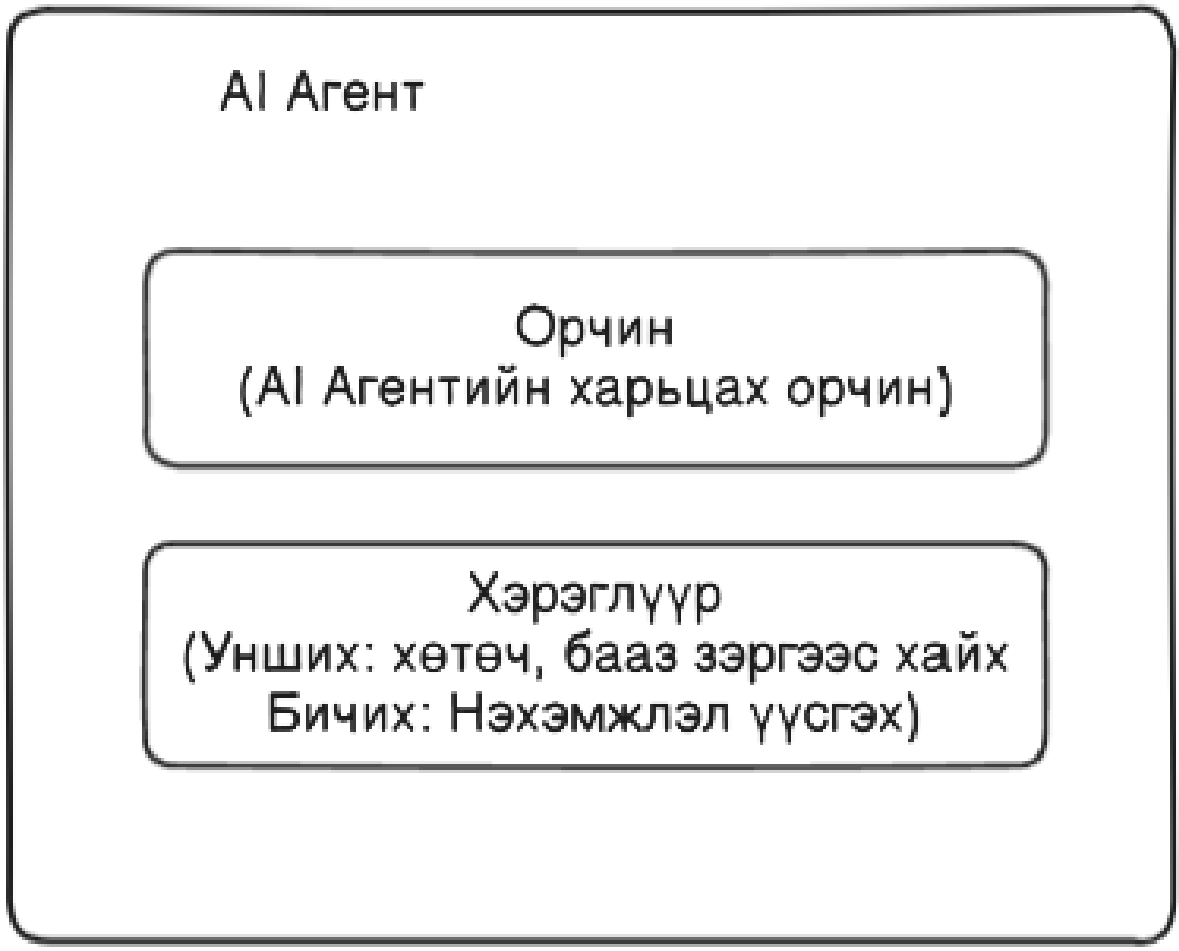
1.3.1 Агент гэж юу вэ

Агент гэдэг нь өөрийн орчныг мэдрэх, түүн дээр үйлдэл хийх чадвартай систем юм. Хиймэл оюунаар дэмжигдсэн агентууд нь суурь моделийн хүч чадлаар дамжуулан бидний туслах, хамтран ажиллагч, сургагч 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.

1.3.2 Агентын бүрэлдэхүүн хэсгүүд

Хиймэл оюун агентыг тодорхойлдог гурван гол зүйл:

1. **Орчин (Environment)**: Агент ажиллах орчин нь түүний хэрэглээний тохиолдлоор тодорхойлогдоно. Жишээ нь: интернэт, гал тогоо, машин(IOT).
2. **Үйлдлүүд (Actions)**: Агентын хийж чадах үйлдлүүд нь түүний хандах боломжтой хэрэглүүрүүдээр өргөжинө.
3. **Даалгавар (Task)**: Хэрэглэгчээс өгөгдсөн даалгавах.



Зураг 1.3: Агентын бүрэлдэхүүн хэсгүүд

1.3.3 Хэрэглүүрүүд (Tools)

Гадаад хэрэглүүр байхгүй бол агентын чадавхи маш хязгаарлагдмал байх болно. Хэрэглүүр нь агентыг илүү чадварлаг болгодог. Хэрэглүүрийн гурван ангилал:

Мэдлэг нэмэгдүүлэх хэрэглүүрүүд

RAG системийн бүрэлдэхүүн хэсгүүд:

- Текст retriever
- Зураг retriever
- SQL executor
- Интернэт хайлт API
- Дотоод хайлтын системүүд

Чадавх өргөтгөх хэрэглүүрүүд

- **Тооны машин**: Хиймэл оюун загварууд математикт сул байдаг. Агентд бэлэн тооны машины API өгснөөр тооцоог оновчтой, хурдан, токены бага зарцуулалтаар гүйцэтгэнэ.
- **Хэрэглэгчийн код уншигч**: Код бичиж, ажиллуулах, үр дүн гаргана. Энэ нь кодчилолын туслах, өгөгдөл шинжлэгч, судалгааны туслах боломжийг олгоно.

Бичих үйлдлийн хэрэглүүрүүд

Зөвхөн унших биш, өөрчлөлт оруулах хэрэглүүрүүд:

- Өгөгдлийн санд өгөгдөл нэмэх, засварлах, устгах
- Мэйл илгээх
- Банкны шилжүүлэг хийх

- Календарт тэмдэглэл нэмэх

Анхааруулга: Бичих үйлдэл нь өндөр эрсдэлтэй. Code injection довтолгооноос болгоомжлох хэрэгтэй.

1.3.4 Төлөвлөлт (Planning)

Төлөвлөлт нь агентын гол үүрэг бөгөөд дараах үе шаттай:

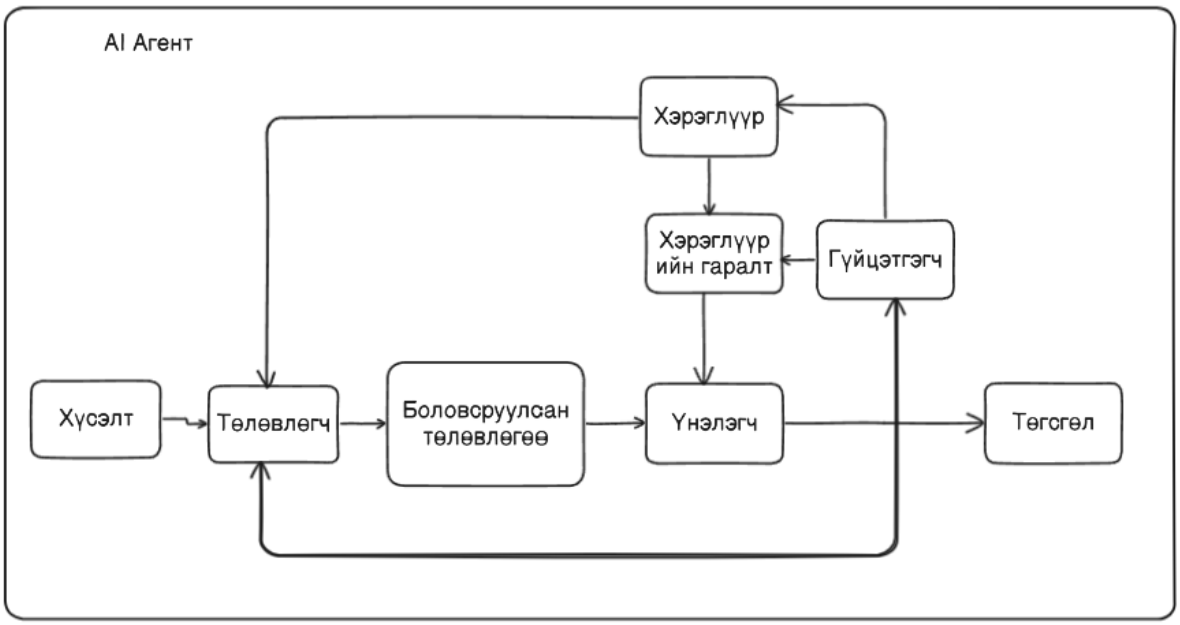
1. **Төлөвлөгөө үүсгэх (Plan Generation)**: Даалгаврыг гүйцэтгэх дараалсан үйлдлүүдийн төлөвлөгөө гаргах. Үүнийг task decomposition буюу даалгаврыг задлах процесс гэж нэрлэнэ.
2. **Эргэцүүлэн бодох ба алдаа засах (Reflection)**: Үүсгэсэн төлөвлөгөөг үнэлэх. Муу байвал шинэ төлөвлөгөө гаргах.
3. **Гүйцэтгэл (Execution)**: Төлөвлөгөөнд заасан үйлдлүүдийг хийх. Энэ нь ихэвчлэн функц дуудах үйлдэл хийнэ.
4. **Үр дүнг үнэлэх**: Үйлдлийн үр дүнг хүлээн авсны дараа зорилго биелсэн эсэхийг тодорхойлох. Алдааг тодорхойлж засах.

1.3.5 Суурь загварууд төлөвлөгч болж чадах уу

Зарим судлаачид LLM нь төлөвлөгч байж чадахгүй гэж үздэг. Учир нь төлөвлөлт нь үндсэндээ хайлтын асуудал бөгөөд autoregressive загвар нь зөвхөн урагш үйлдэл үүсгэж чаддаг гэж үздэг. Гэвч бодит байдал дээр загвар дахин эхлэж өөр зам сонгож чаддаг тул сургалт сайн хийснээр сайн төлөвлөгөө гарч болно.

Төлөвлөлтийг сайжруулах аргууд:

- Илүү сайн system prompt бичих, жишээ олноор өгөх
- Хэрэглүүрүүдын тайлбарыг илүү сайн бичих
- Функцүүдийг хялбарчлах, задлах



Зураг 1.4: Агентын төлөвлөлт

- Илүү хүчтэй загвар ашиглах
- Төлөвлөлтөд зориулж загвар нарийвчлан сургах

Мөн хэрэглүүрийг нэмэх нь fine-tuning хийснээс илүү үр дүнтэй гэж үзжээ.

1.3.6 Эргэцүүлэн бодох ба алдаа засах

Хамгийн сайн төлөвлөгөө байнга үнэлэгдэж, тохируулагдах шаардлагатай. Reflection нь агентын амжилтад чухал үүрэг гүйцэтгэнэ. Үүнийг хоёр аргаар хийж болно:

- **Self-critique**: Ижил загвар өөртэйгөө ярилцаж алдааг илрүүлнэ.
- **Тусдаа үнэлэгч**: Тусдаа загвар эсвэл функц үр дүнд оноо өгнө.

1.3.7 Агентын санах ой

Хиймэл оюун загвар нь гурван санах ойн механизмтай:

1. **Дотоод мэдлэг (Internal Knowledge)**: Загвар өөрөө нь дотоод мэдээлэлтэй. Сургалтын өгөгдлөөс олж авсан мэдлэг. Моделийг шинэчлэхгүй бол өөрчлөгдөхгүй.

2. **Богино хугацааны санах ой (Short-term Memory)**: Моделийн контекст. Өмнөх мессежүүд контекстэд нэмэгдэж болно. Даалгавар дууссаны дараа устдаг. Хурдан боловч багтаамж хязгаарлагдмал.

3. **Урт хугацааны санах ой (Long-term Memory)**: Гадаад өгөгдлийн эх сурвалж (RAG). Даалгаврууд хооронд үргэлжилдэг. Өгөгдлийг устгаж болно.

1.3.8 Агентын дизайны загварууд

Агентууд нь зөвхөн үндсэн чадваруудаасаа биш, тэдгээрийн workflow болон харилцааг бүтэцлэх зохиомжийн загваруудаас ч ажил авдаг. Эдгээр загварууд нь агентуудад нарийн төвөгтэй асуудлыг шийдэх, хувьсах орчинд дасан зохицох, үр дүнтэй хамтран ажиллах боломжийг олгодог. Жишээлбэл үнэтэй моделийг үнэтэй процессд үлдээгээд, хямдхан GPT-2 мэтийн моделээр асуултын чиглүүлэг хийж болно. Иймээс мульти-агент байх нь практикт тохиромжтой элбэг арга зам юм.

Эргэцүүлэн бодох (Reflection)

Reflection 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.

Хэрэглүүрийн хэрэглээ (Tool Use)

Гадаад хэрэглүүрүүдтэй холбогдох нь агентын функционалыг өргөжүүлж, өгөгдөл авах, үйл явцыг автоматжуулах, эсвэл тодорхой workflow-г гүйцэтгэх зэрэг даалгаврыг гүйцэтгэх боломжийг олгоно. Энэ нь математик тооцоолол эсвэл өгөгдлийн сангийн асуулга гэх мэт нарийвчлал шаардлагатай үйлдлүүдэд онцгой ач холбогдолтой.

Хэрэглүүрийн хэрэглээ нь уян хатан шийдвэр гаргалт болон таамагладаг, найдвартай гүйцэтгэлийн хоорондох хоосон зайг нөхдөг.

Төлөвлөлт (Planning)

Төлөвлөлтийн чадвартай агентууд нь өндөр түвшний зорилгуудыг үйлдэл хийх боломжтой алхмуудад задалж, даалгавруудыг логик дарааллаар зохион байгуулж чаддаг. Энэ дизайны загвар нь олон алхам бүхий асуудлыг шийдэх эсвэл хамааралтай workflow-г удирдахад чухал юм.

Жишээлбэл, аялал төлөвлөх агент нь дараах төлөвлөгөө гаргаж болно:

1. Нислэг хайх
2. Зочид буудал захиалах
3. Үзвэр сервисийн цэгүүдийг судлах
4. Өдөр бүрийн маршрут үүсгэх

Олон агентын хамтын ажиллагаа (Multi-Agent Collaboration)

Олон агентын системүүд нь асуудлыг шийдэхэд модуляр арга барил ашигладаг. Иймээс тодорхой даалгавруудыг мэргэшсэн агентуудад хуваарилдаг. Энэ арга нь уян хатан байдлыг санал болгодог. Мөн үр ашигтай байдлыг сайжруулахын тулд даалгаварт агентуудад илүү жижиг хэлний моделууд (SLMs) ашиглаж болно.

Модульчлагдсан зохиомж нь тэдгээрийн контекстыг тусгай даалгаврууддаа чиглүүлэх замаар агент тус бүрийн нарийн төвөгтэй байдлыг багасгадаг. Хамтран ажиллахаар эдгээр 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.

1.4 Хиймэл оюун инженерчлэлийн хэрэгжүүлэлт

Суурь загварууд ашиглан аппликейшн хөгжүүлэх нь уламжлалт ML инженерчлэлээс гурван чухал талаараа ялгаатай:

1. **Моделийн дасан зохицуулалт**: Өөрөө модел сургахын оронд бусдын сургасан моделийг ашиглана. Иймээс уг моделийг дасан зохицуулах нь чухал болсон.

2. **Optimization**: Суурь модел нь илүү их тооцоолол шаарддаг, хоцрогдол өндөр. Үр ашигтай сургалтын төлөө inference optimization хийснээр бага хүчтэй компьютерээр, бага хугацаанд гүйцэтгүүлж болно. Иймээс бэлэн суурь моделийн API ашиглах уу, эсвэл өөрөө агентээ серверт байршуулах уу гэдгийг шийдэж болно.

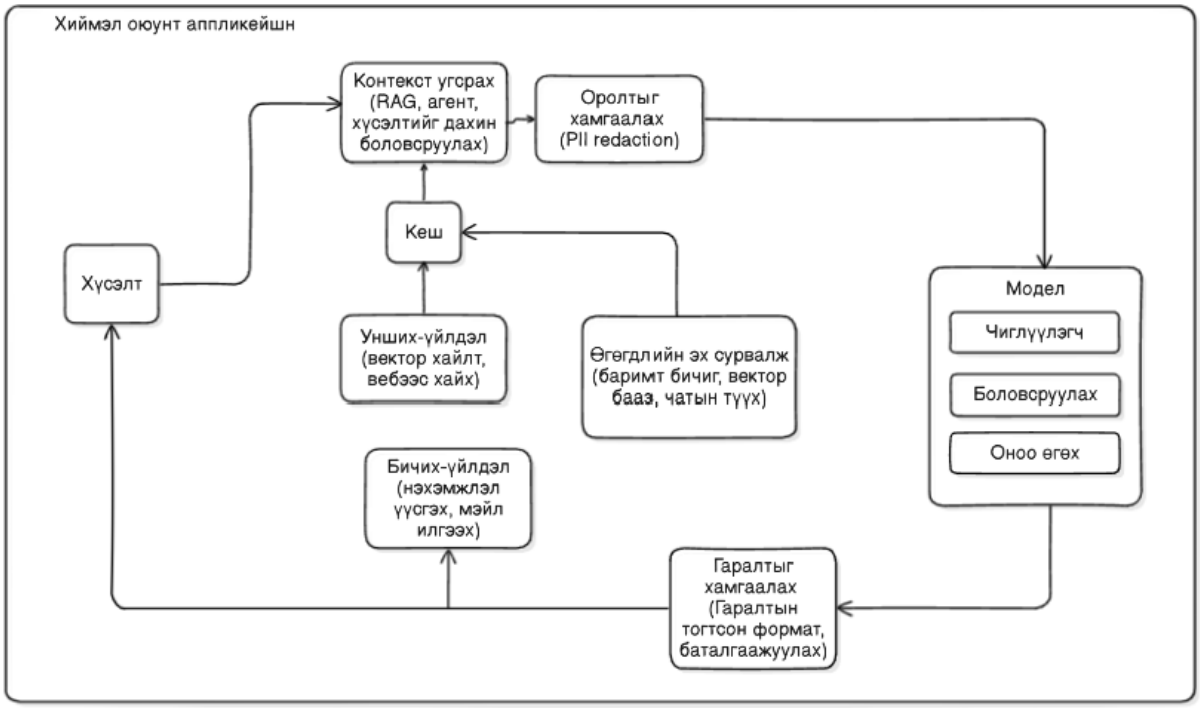
3. **Нээлттэй гаралт**: Загварууд нээлттэй төгсгөлтэй гаралт үүсгэдэг тул үнэлгээ хийхэд илүү бэрхшээлтэй. Дундах процессийг мэдэж болдоггүй.

Загвар дасан зохицуулалтын хоёр гол арга:

- **Prompt-based techniques**: Моделийн жин өөрчлөхгүйгээр зааварчилгаа, контекст өгч дасан зохицуулна. Хялбар, цөөн өгөгдөл шаардана. Prompt engineering энд хамаарна.

- **Finetuning**: Моделийн жинг өөрчилж дасан зохицуулна. Илүү нарийн төвөгтэй, илүү их өгөгдөл шаардана. Гэвч чанар, хоцрогдол, өртгийг мэдэгдэхүйц сайжруулж чадна.

- **Хэрэглүүр өгөх**: Хэрэглүүр өгснөөр уг даалгаврыг маш онончтой, хоцрогдол багатайгаар, зардлын өртгийг мэдэгдэхүйц сайжруулж чадна.



Зураг 1.5: Хиймэл оюуны аппликейшн

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 үйлдлүүд нэг том, нягт нэгдсэн кодын санд амьдардаг байв. Монолитууд нь хөгжүүлэх, нэвтрүүлэлт хийхэд энгийн байсан.

2.1.2 Монолитыг өргөжүүлэх сорилт

Гэвч аппликейшнууд томрох тусам асуудлууд нэмэгдсэн. Монолитыг өргөжүүлэх нь бүх зүйлийг нэгтгэсэн байдлаар өргөжүүлдэг. Нэг модульд хийсэн жижиг өөрчлөлт кодын санд орсноор энэ нь шинэчлэлтийг удаашруулж, унахад эрсдэлтэй болгодог. Өөр өөр функц дээр ажиллаж байгаа багууд байнга бие биенийхээ кодод 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 болов.

2.1.3 Микросервис рүү шилжих

Эдгээр хязгаарлалтаас ангижрахын тулд компаниуд микросервис архитектур руу шилжүүлсэн. Энэ өөрчлөлт нь багуудад салангид байдлаар өөрчлөлтийг хурдан deploy хийх, аппликейшнийг дахин deploy 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.

Микросервисийн гол онцлогууд:

- **Бие даасан байдал**: Микросервис бүр бие даасан үүрэгтэй, өөрийн өгөгдлийн сантай.
- **Уян хатан хөгжүүлэлт**: Өөр өөр багууд өөр өөр технологи, хэл ашиглаж хөгжүүлж болно.
- **Өргөжих чадвар**: Зөвхөн шаардлагатай сервисийг л өргөжүүлж болно.
- **Найдвартай байдал**: Нэг сервис унавал бусад сервисүүд ажиллаж үргэлжлэнэ.

2.2 Микросервисийн давуу тал

1. **Технологийн олон янз байдал**: Сервис бүр өөрийн хэрэгцээнд тохирсон технологи сонгож болно. Жишээ нь: нэг сервис Python, нөгөө нь Go, өөр нь Node.js ашиглаж болно.
2. **Багуудын бие даасан ажиллагаа**: Баг бүр өөрийн сервисийг бие даан хөгжүүлж, deploy хийж чадна.
3. **Хурдан deployment**: Том системийг бүхэлд нь дахин deploy хийх шаардлагагүй. Зөвхөн өөрчлөгдсөн сервисийг л deploy хийнэ.
4. **Илүү сайн өргөжих чадвар**: Ачаалал их сервисийг л өргөжүүлэх нь бүх системийг өргөжүүлэхээс үр ашигтай.
5. **Алдааны тусгаарлалт**: Нэг сервисийн алдаа бусад сервисд дамжихгүй.

2.3 Микросервисийн сорилтууд

Микросервис архитектур олон давуу талтай ч дараах сорилтуудтай:

1. **Нарийн төвөгтэй байдал**: Олон сервисүүдийг удирдах, тэдгээрийн харилцааг хянах нь монолит системээс илүү төвөгтэй.
2. **Өгөгдлийн тогтмол байдал**: Сервис бүр өөрийн өгөгдлийн сантай байх нь өгөгдлийн нийцтэй, уялдаатай байдлыг хангахад хэцүү болгодог.
3. **Сүлжээний хоцрогдол**: Сервисүүд хоорондоо сүлжээгээр харилцах нь нэмэлт хоцрогдол авчирна.
4. **Алдаа илрүүлэх хэцүү байдал**: Олон сервисийгээр дамжин явах хүсэлтийн алдааг олох, засах хэцүү.
5. **Сервис хоорондын харилцаа**: Сервисүүд хэрхэн харилцах, мэдээлэл солилцох нь нарийн асуудал.
6. **Transaction удирдлага**: Олон сервисийгээр transaction явуулах нь өгөгдлийн сан түгжигдэж асуудал гарах эрсдэлтэй.

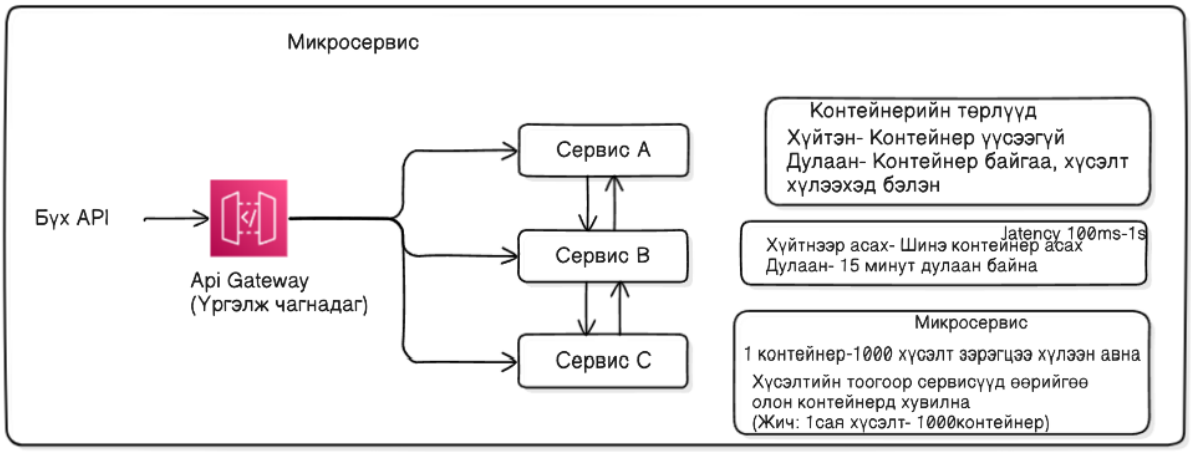
2.4 Микросервис хоорондын харилцаа

Микросервисүүд хоорондоо хоёр гол аргаар харилцдаг:

2.4.1 Синхрон харилцаа (Synchronous Communication)

HTTP/REST API эсвэл gRPC ашиглан шууд хүсэлт илгээж хариу хүлээнэ. Энэ нь хялбар боловч:

- Tight coupling үүсгэдэг
- Нэг сервис унавал бусад сервисүүд алдаа гаргана
- Latency нэмэгддэг



Зураг 2.1: Синхрон микросервис

NxM-ээр сервисийн холболт нэмэгдэх учир үүнийг удирдахад маш хүндрэлтэй болно.

2.4.2 Асинхрон харилцаа (Asynchronous Communication)

Message queue (RabbitMQ, Kafka) ашиглан мессеж солилцоно. Энэ нь:

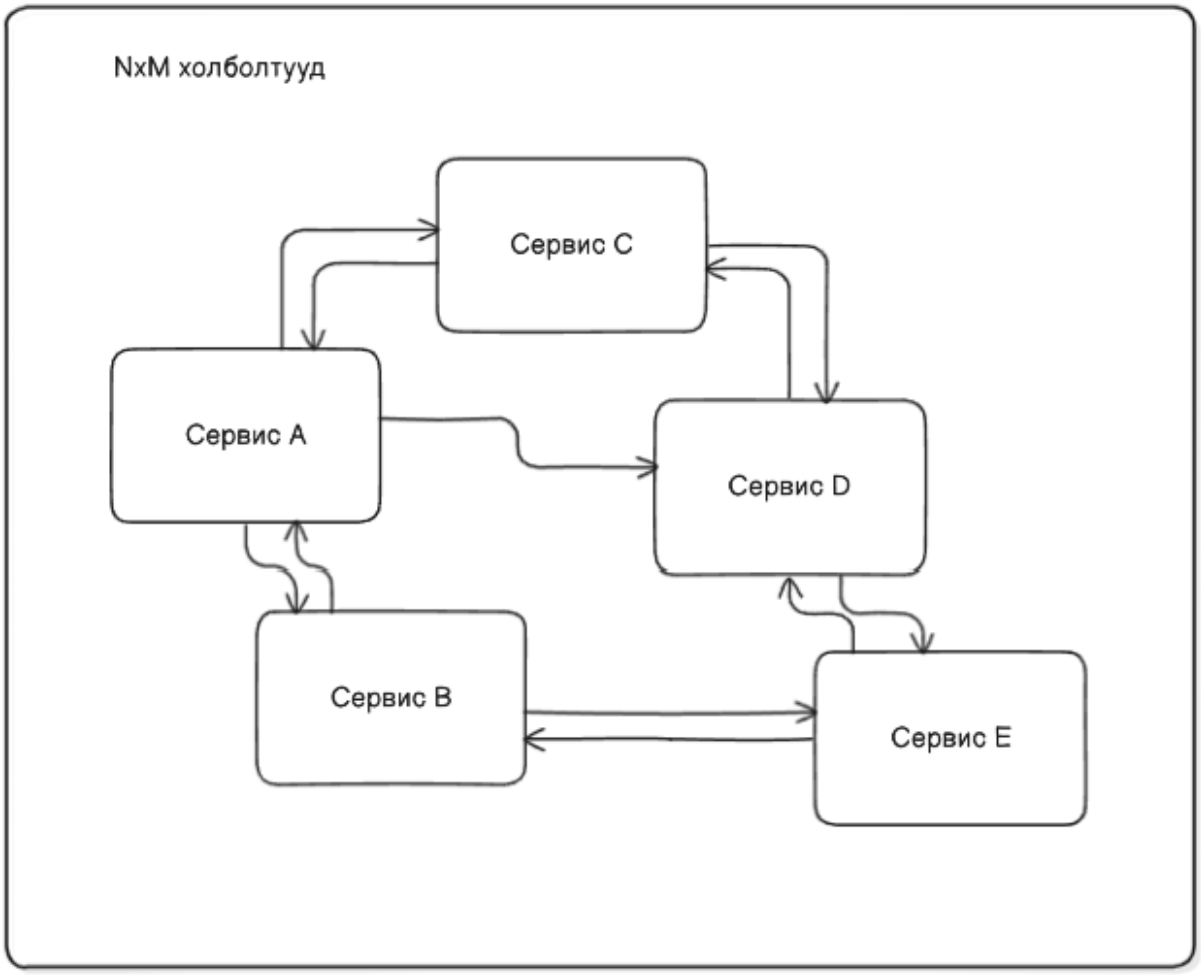
- Loose coupling үүсгэнэ
- Илүү найдвартай
- Илүү төвөгтэй

2.5 Event-Driven архитектур

2.5.1 Event-Driven архитектур гэж юу вэ

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

Үйл явдлаар удирдагдах архитектур (Event-Driven Architecture, EDA) нь энэ асуудлыг шийдсэн. Нягт холбогдсон, синхрон харилцааны оронд EDA нь бүрэлдэхүүн хэсгүүдэд

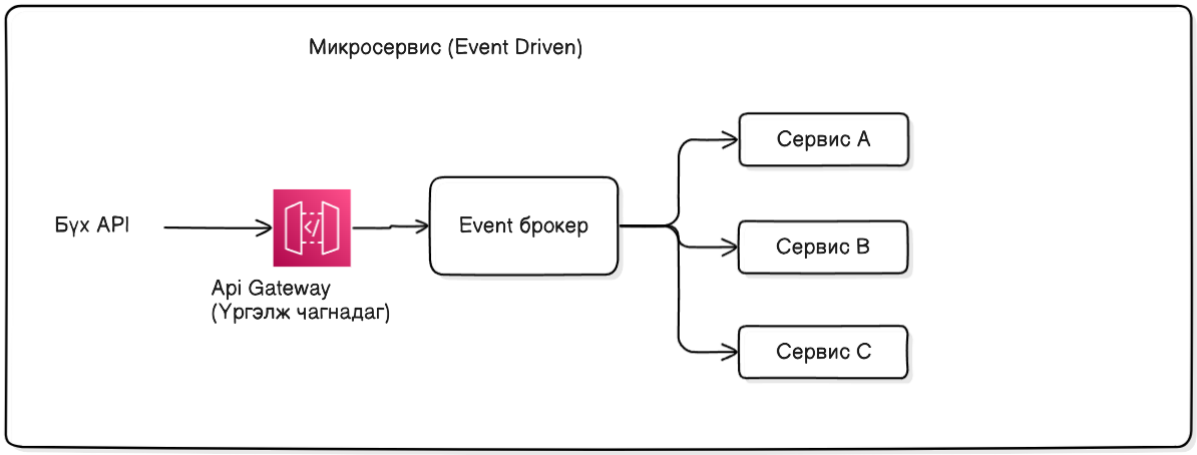


Зураг 2.2: Синхрон микросервисийн NxM холбоо

үйл явдлаар асинхрон харилцах боломжийг олгодог. Сервисүүд бие биенийхээ хүлээхгүй. Бодит цагт юу болж байгааг мэдээд хариу үйлдэл үзүүлдэг.

2.5.2 EDA-ын давуу тал

1. **Салангид байдал (Decoupling)**: Сервисүүд үйл явдлаар харилцдаг тул тэд бие биенээсээ хараат бус байна. Нэг сервис өөрчлөгдөх эсвэл унах нь бусад сервисд шууд нөлөөлөхгүй.
2. **Өргөжүүлэх чадвар**: Сервис бүр үйл явдлыг бие даан боловсруулах тул системийг өргөжүүлэх илүү хялбар.



Зураг 2.3: Асинхрон микросервис

3. **Уян хатан байдал**: Шинэ сервис нэмэх эсвэл одоо байгаа сервисийг өөрчлөх нь бусад сервисд өөрчлөлт шаардахгүй.

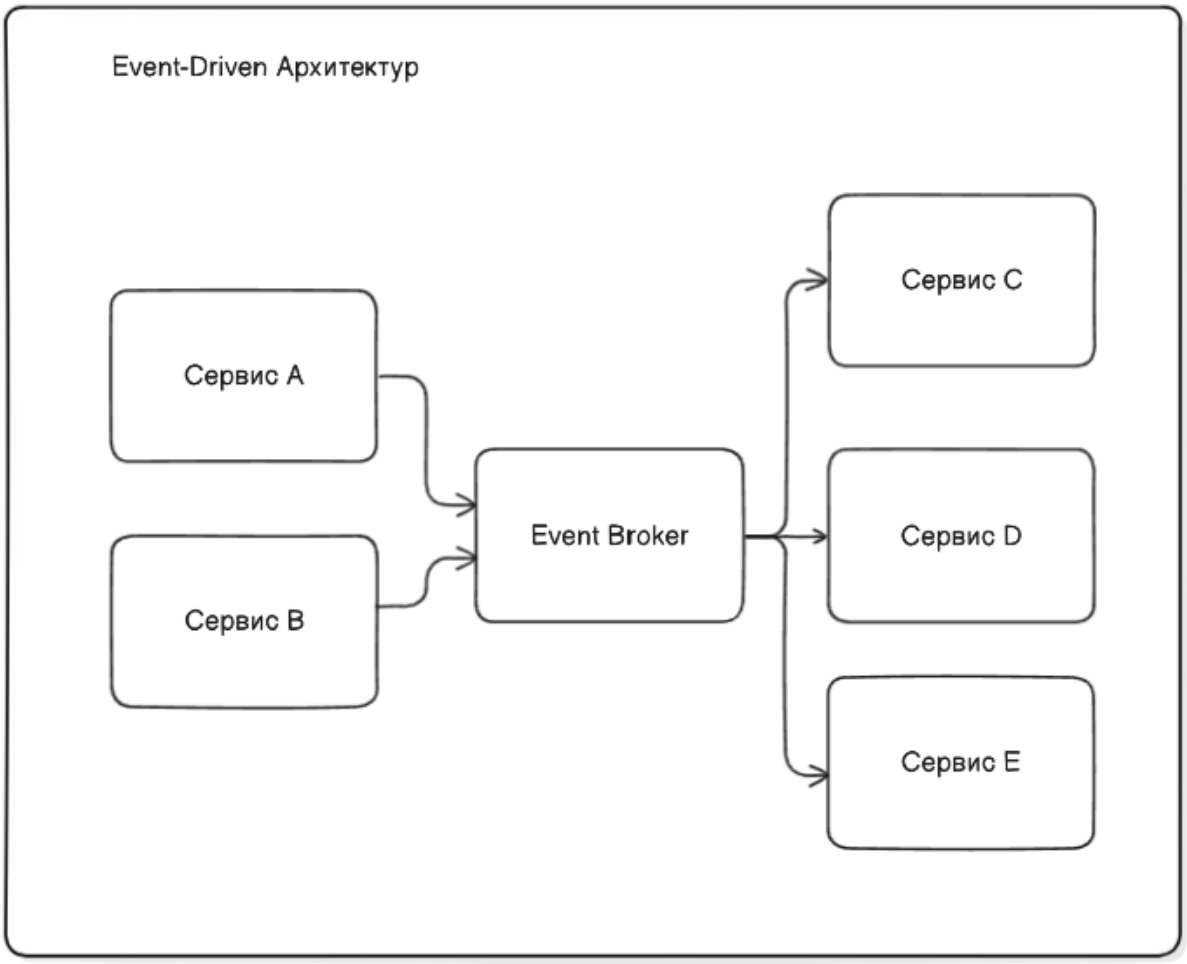
4. **Бодит цагийн боловсруулалт**: Үйл явдлууд тэр даруйдаа боловсруулагдах тул систем нь өөрчлөлтөд хурдан хариу үйлдэл үзүүлнэ.

2.5.3 Apache Kafka: Event-Driven архитектурын хүчирхэг суурь

Apache Kafka нь салангид, өндөр дамжуулалттай, бага хоцрогдолтой Event-Driven архитектурын төв мэдрэлийн систем болж чаддаг.

Kafka-ийн үндсэн ойлголтууд

- **Topic**: Үйл явдлуудыг ангилах логик channel. Жишээ нь: ”user-events”, ”order-events”
- **Producer**: Үйл явдлыг topic руу бичдэг аппликейшн
- **Consumer**: Topic-оос үйл явдлыг уншдаг аппликейшн
- **Partition**: Topic-ийг өргөжүүлэх, параллель боловсруулалт хийх боломжтой болгодог
- **Consumer Group**: Олон consumer нэг багаар ажиллаж ачааллыг хуваарилдаг



Зураг 2.4: Асинхрон микросервис

Kafka-ийн давуу тал

1. **Хэвтээ өргөжих чадвар**: Kafka-ийн салангид зохиомж нь саадгүйгээр шинэ агент эсвэл consumer нэмэх боломжийг олгоно.

2. **Бага хоцрогдол**: Бодит цагийн үйл явдал боловсруулалт нь агентуудад өөрчлөлтөд шууд хариу үйлдэл үзүүлэх боломжийг олгоно.

3. **Loose Coupling**: Topic-уудаар харилцах нь агентууд хараат бус, өргөжүүлэх боломжтой байхыг баталгаажуулна.

4. **Үйл явдлын хадгалалт**: Тогтвортой мессежийн хадгалалт нь өгөгдөл дамжилтын явцад алга болохгүй гэдгийг баталгаажуулна.

5. **Дахин тоглуулах боломж (Replayability)**: Kafka нь үл хөдлөх салангид log учраас үйл явдал бүр хадгалагдаж, debug, үнэлгээ, дахин сургалтад дахин тоглуулж болно.

2.5.4 Apache Flink: Streaming боловсруулалтын хөдөлгүүр

Apache Flink нь stateful тооцоолол, үйл явдал цагийн боловсруулалт хийх чадвартай салангид stream processing framework юм. Flink нь Kafka-тай хамт ашиглах замаар хүчирхэг бодит цагийн өгөгдөл боловсруулалтын конвейер бүтээх боломжийг олгоно.

Flink-ийн давуу тал

- **Stateful боловсруулалт**: Flink нь state-г найдвартай удирдаж, нарийн төвөгтэй тооцоолол хийх боломжийг олгоно.
- **Бодит цагийн боловмруулалт**: Үйл явдлын цаг дээр үндэслэн бодит цагийн шинжилгээ хийх.
- **Өндөр дамжуулалт**: Секундэд сая сая үйл явдлыг боловсруулж чаддаг.
- **Exactly-once семантик**: Мэдээлэл нэг удаа л боловсруулагдахыг баталгаажуулна.

Flink AI Inference

Flink нь LLM-тэй ажиллах чадвартай. Flink AI inference нь өгөгдлийг авч, LLM рүү илгээж, хариу авах боломжийг олгоно. Энэ нь төлөвлөгч агентыг Flink app болгон хөгжүүлэх боломжийг олгодог.

Жишээлбэл:

- Kafka topic-оос үйл явдал уншина
- LLM ашиглан контекст ойлгох, төлөвлөгөө гаргах
- Үр дүнг өөр Kafka topic руу бичих
- Бусад агентууд энэ үр дүнг уншиж гүйцэтгэнэ

Flink болон RAG

Hallucination-ийг багасгахын тулд LLM-ийг бодит өгөгдөлд суурилуулах хэрэгтэй. Flink нь RAG pattern-ийг бүтээхэд тусална.

1. Flink нь өгөгдлийг боловсруулна
2. LLM inference ашиглан өгөгдлийг embedding болгон хөрвүүлнэ
3. Embedding-ыг Kafka topic руу бичнэ
4. Kafka Connect ашиглан vector database руу синхрончлоно
5. Агентууд RAG ашиглан бодит өгөгдөл дээр суурилсан хариулт өгнө

2.5.5 Агентууд ба EDA

Хиймэл оюун агентуудыг өргөжүүлэх нь үндсэндээ салангид системийн асуудал юм. Агентууд нь шийдвэр гаргаж, үйлдэл хийхийн тулд олон эх сурвалж, бусад агентууд, хэрэглүүрүүд, гадаад системүүдээс мэдээлэл цуглуулах шаардлагатай.

Агентууд яагаад EDA шаарддаг:

- **Асинхрон шинж чанар**: Агентууд нь хүн шиг ажилладаг. Агент нь олон эх сурвалжаас мэдээлэл цуглуулж, өгөгдлийг шинжилж, үйлдэл хийхээсээ өмнө шийдвэр гаргах хэрэгтэй. Эдгээр үйл явцууд нь асинхрон шинжтэй.
- **Мэдээллийн хамаарал**: Микросервисүүд ихэвчлэн салангид үйлдлүүдийг боловсруулдаг бол агентууд нь контекст баялаг, хуваалцсан мэдээлэлд найддаг. Энэ нь хамаарлыг удирдах, бодит цагийн өгөгдлийн урсгалыг баталгаажуулах онцгой шаардлагыг бий болгоно.
- **Олон хэрэглэгчдэд үйлчлэх**: Агентын гаралт нь зөвхөн AI app рүү биш, тэд өгөгдлийн сан, CRM, CDP, customer success платформ зэрэг бусад чухал системүүд рүү хандаж, лог хадгалж болно.

Агентуудыг RPC болон API-аар холбож болно, гэхдээ энэ нь нягт холбогдсон системүүд бий болгодог. Энэхүү нягт холбоос нь өргөжүүлэх, дасан зохицох, эсвэл ижил өгөгдлийн олон хэрэглэгчдийг дэмжихэд хэцүү болгодог. Агентууд нь уян хатан байдал шаарддаг. Тэдний гаралт нь бусад агентууд, сервисүүд, платформуудад хатуу хамааралгүйгээр үр дүнгүй дамжих ёстой.

3. АСУУДЛЫН ТОДОРХОЙЛОЛТ

3.1 Микросервис архитектур дахь сорилтууд

Микросервис архитектурт дараах гол асуудлууд байдаг.

3.1.1 Сервис хоорондын нарийн төвөгтэй логик

Хэрэглэгчийн нэг хүсэлт нь олон сервисийгээр дамжин явах ёстой. Жишээ нь онлайн худалдааны систем дээр:

1. Хэрэглэгч бараа захиална
2. Захиалгын сервис захиалга үүсгэнэ
3. Агуулахын сервисээс бараа байгаа эсэхийг шалгана
4. Төлбөрийн сервисийгээр төлбөр хийнэ
5. Хүргэлтийн сервисд мэдэгдэнэ
6. Notification сервисийгээр хэрэглэгчид мэдэгдэнэ

Эдгээр алхам бүрт алдаа гарч болох бөгөөд алдаа гарах үед юу хийх нь тодорхойгүй байдаг. Уламжлалт аргаар энэ workflow-г хатуу кодлох нь:

- Уян хатан бус
- Шинэ сервис нэмэхэд хэцүү
- Алдаа засалт хэцүү
- Бизнес дүрмийн өөрчлөлт хийхэд том код өөрчлөлт шаардлагатай

3.1.2 Өгөгдлийн нэгтгэл ба шинжилгээ

Микросервис бүр өөрийн өгөгдлийн сантай байх нь:

- Олон өгөгдлийн сангийн өгөгдлийг нэгтгэх хэцүү
- Cross-service шинжилгээ хийх төвөгтэй
- Тайлангийн системд өгөгдөл цуглуулах асуудалтай
- Хэрэглэгчийн бүрэн дүр төрхийг харах хэцүү

3.1.3 Динамик routing ба шийдвэр гаргалт

Хэрэглэгчийн хүсэлтийг аль сервис рүү чиглүүлэх, хэдэн сервисийг дуудах шаардлагатайг тодорхойлох нь:

- Бизнес логик өөрчлөгдөх бүрд код өөрчлөх шаардлагатай
- A/B testing хийхэд төвөгтэй
- Орчны өөрчлөлтөд дасан зохицох чадваргүй

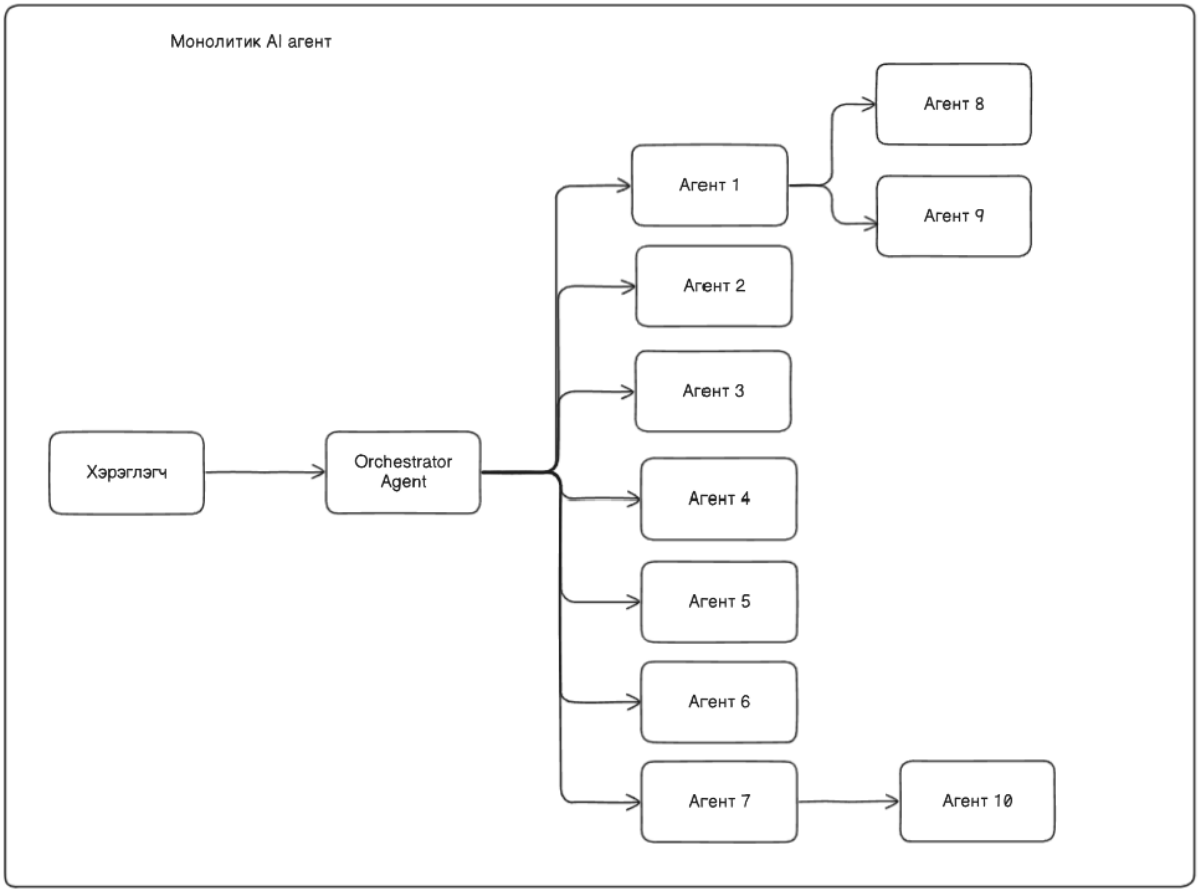
3.1.4 Мониторинг ба алдаа илрүүлэлт

- Олон сервисийн логийг нэгтгэх хэцүү
- Хэрэглэгчийн хүсэлт аль сеовисд алдаа гаргасныг тодорхойлох хэцүү
- Алдааны язгуур шалтгааныг олох төвөгтэй

3.1.5 Монолит Агент

Агент суурилсан систем хөгжүүлэхдээ анхны хувилбар нь ихэвчлэн монолит хэлбэртэй байдаг. Controller, Planner, SQL Handler, Streaming Handler зэрэг агентууд нэг аппликейшн дотор ажилладаг. Энэ арга нь эхэндээ ажилладаг ч дараах асуудлуудтай:

- **Deployment хамаарал**: Planner-ийн v2 гаргахын тулд бүх системийг дахин deploy хийх шаардлагатай.



Зураг 3.1: Монолит Агент

- **Hardware хязгаарлалт**: Planner агент нь LLM ашиглаж GPU шаарддаг бол SQL Handler нь жижиг SLM ашиглан CPU дээр ажиллаж болно. Монолит систем энэ хоёрыг салгаж чадахгүй.
- **Өргөжүүлэх боломжгүй**: Нэг агентад их ачаалал ирсэн ч бүх системийг өргөжүүлэх шаардлагатай болно.
- **Нягт холбоос**: Агентууд хоорондоо шууд дуудлагаар холбогдсон байх нь алдааны дамжлага, хоцрогдлын асуудал бий болгоно.

3.2 Хиймэл оюун агентууд хэрхэн туслах вэ

Хиймэл оюун агентууд нь дээрх асуудлуудыг дараах аргаар шийдэж чадна:

1. **УIntelligent Orchestrator**: Агент нь хүсэлтийг ойлгож, аль сервисүүдийг дуудах хэрэгтэй, ямар дарааллаар дуудах, алдаа гарвал яах талаар төлөвлөгөө гарган гүйцэтгэнэ.

2. **Контекст бүтээгч (Context Builder)**: RAG ашиглан олон өгөгдлийн сангийн мэдээллийг нэгтгэж, ойлгомжтой контекст бүтээнэ.

3. **Динамик routing**: Хэрэглэгчийн хүсэлтийг ойлгож, хамгийн тохиромжтой сервис рүү чиглүүлнэ.

4. **Data Analyst**: Олон сервисийн өгөгдлийг цуглуулж, шинжилж, ойлгомжтой тайлан гаргана.

5. **Алдаа засагч (Error Handler)**: Алдаа гарсан тохиолдолд шалтгааныг олж, засах төлөвлөгөө санал болгоно.

3.3 Event-Driven архитектурын шаардлага

Агентуудыг үр дүнтэй өргөжүүлэхийн тулд тэдгээрийг микросервис болгон салгах шаардлагатай. Гэхдээ микросервисүүд нь RPC эсвэл API дуудлагаар харилцах нь дахин монолиттой адилхан асуудал үүсгэнэ.

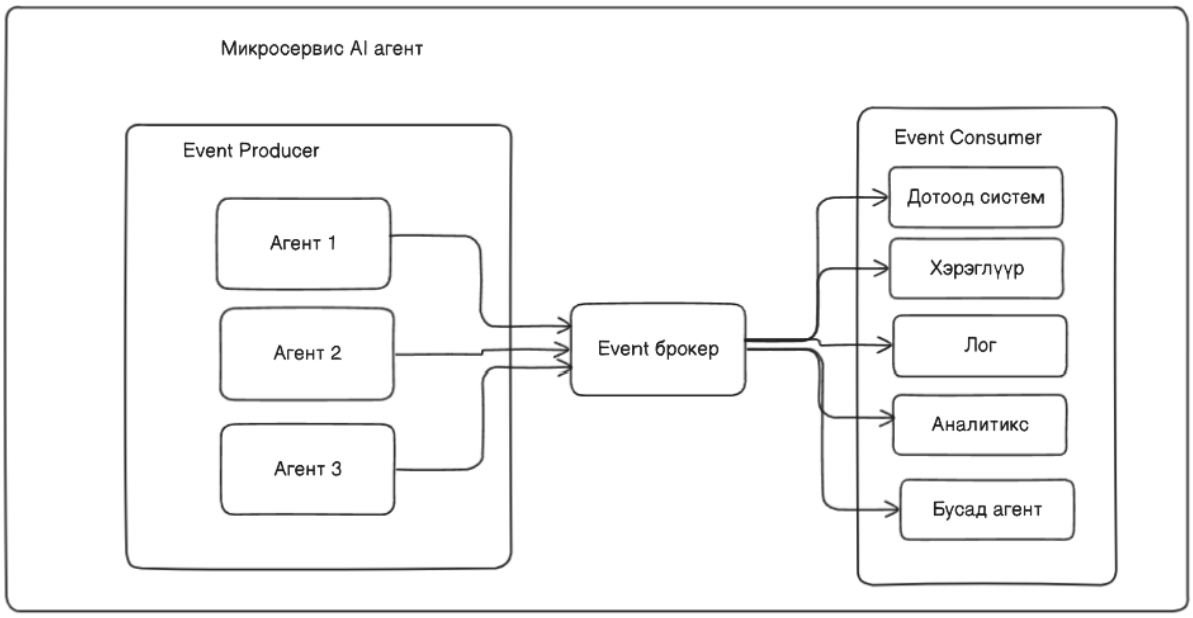
Event-Driven архитектур (EDA) нь дараах асуудлуудыг шийднэ:

- **Салангид байдал**: Агентууд хоорондоо Kafka topics-оор харилцах тул шууд хамаарал байхгүй.
- **Асинхрон**: Агентууд бие биенийхээ хүлээхгүй, бодит цагт хариу үйлдэл үзүүлдэг.
- **Олон хэрэглэгч**: Нэг агентын гаралтыг олон агент, систем ашиглаж болно.
- **Өргөжих чадвар**: Агент бүр бие даан өргөжиж болно.
- **Найдвартай байдал**: Kafka-ийн үйл явдлын хадгалалт нь мэдээлэл алдагдахгүйг баталгаажуулна.

4. ШИЙДЭЛ БА САНАЛ БОЛГОЖ БУЙ ЗАГВАР

4.1 Хиймэл оюун агент суурилсан микросервис архитектур

Энэхүү судалгаанд микросервис архитектурт хиймэл оюун агентуудыг нэвтрүүлэх моделийг санал болгож байна. Уг загвар нь дараах гол бүрэлдэхүүнтэй:



Зураг 4.1: Хиймэл оюун агент суурилсан микросервис

4.1.1 Агентын архитектур

Orchestrator Agent

Энэ агент нь хэрэглэгчийн хүсэлтийг хүлээн авч, ямар сервисүүдийг дуудах, ямар дарааллаар дуудах талаар төлөвлөгөө гаргадаг. Үүний бүтэц:

- **Intent Understanding**: LLM ашиглан хэрэглэгчийн хүсэлтийг ойлгох
- **Service Discovery**: Боломжтой сервисүүдийн жагсаалтыг мэдэх

- **Execution Planning**: Дуудах сервисүүдийн дараалал, параметрүүдийг тодорхойлох
- **Error Handling**: Алдаа гарвал алдааг засах төлөвлөгөө гаргах

Service Agents

Сервис бүр өөрийн агенттай байж болно. Энэ агент нь:

- Сервисийн API-г тайлбарлаж өгнө
- Хүсэлтийг тухайн сервисд тохирсон форматруу хөрвүүлнэ
- Үр дүнг ойлгомжтой хэлбэрт хөрвүүлнө
- Сервисийн статус, чадавхийг мэдээлнэ

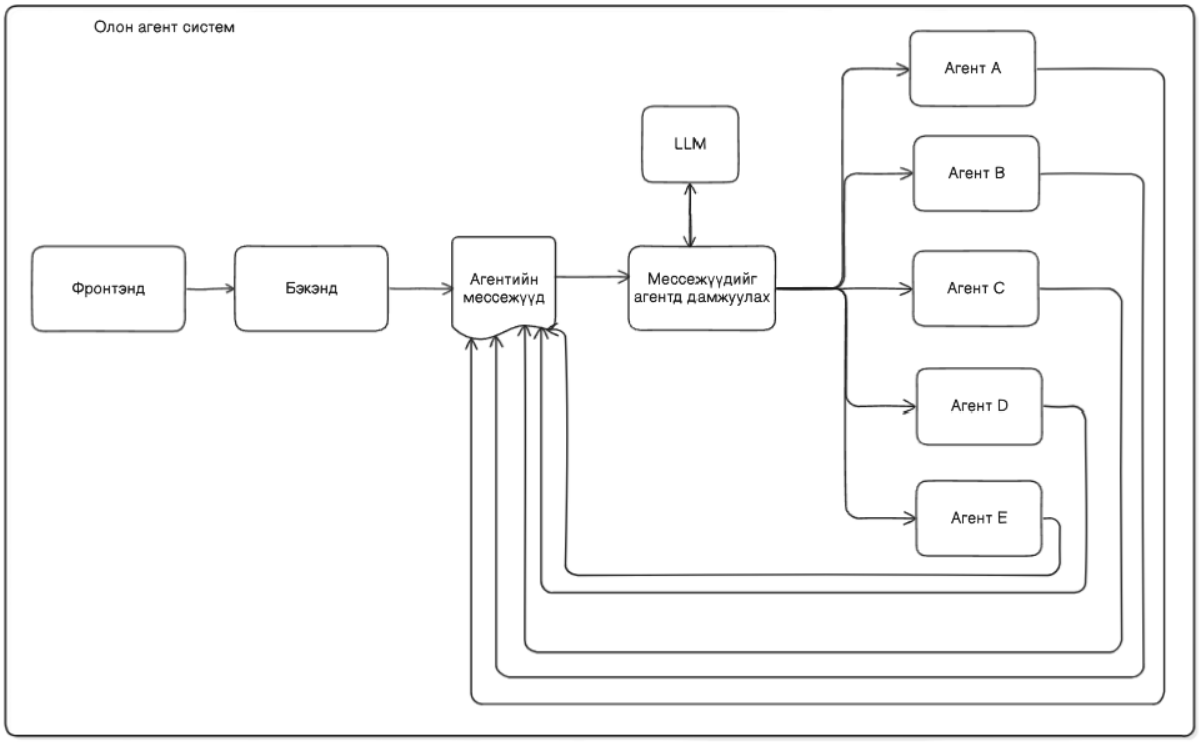
Knowledge Agent

RAG систем ашиглан:

- Сервисүүдийн баримт бичгийг хадгална
- Өмнөх хүсэлт, хариултуудыг санана
- Бизнес дүрэм, журмуудыг мэднэ
- API спецификацийг хадгална

Monitoring Agent

- Логуудыг цуглуулж, шинжилнэ
- Алдаануудыг илрүүлнэ
- Системийн эрүүл байдлыг хянана
- Сайжруулах санал өгнө



Зураг 4.2: Хиймэл оюун агент суурилсан микросервис Event-Driven архитектур

4.1.2 Системийн архитектур

Санал болгож буй Event-Driven архитектур нь:

4.1.3 Kafka Topics ба хамаарал

Системд дараах Kafka topics ашиглагдана:

- **user-requests**: Хэрэглэгчийн хүсэлтүүд
- **planning-tasks**: Төлөвлөгөө гаргах даалгаврууд
- **execution-plans**: Гүйцэтгэх төлөвлөгөөнүүд
- **knowledge-queries**: RAG хайлтын асуултууд
- **knowledge-results**: RAG-ийн үр дүнгүүд
- **service-calls**: Микросервис дуудлагууд

- **service-results**: Микросервисийн үр дүнгүүд
- **user-responses**: Хэрэглэгчид буцах хариултууд
- **monitoring-events**: Системийн логууд ба мониторинг

4.1.4 Event-Driven Workflow

1. **Хүсэлт publish**: API Gateway хүсэлтийг `user-requests` topic руу бичнэ
2. **Orchestrator consume**: Orchestrator Agent үйл явдлыг уншиж, intent-ийг ойлгоно
3. **Төлөвлөлт үйл явдал**: Хэрэв төлөвлөлт хэрэгтэй бол `planning-tasks` topic руу үйл явдал илгээнэ
4. **Flink Planning**: Planner Agent (Flink app) үйл явдлыг уншиж, LLM ашиглан төлөвлөгөө гаргаж, `execution-plans` topic руу бичнэ
5. **Knowledge хайлт**: Orchestrator нь `knowledge-queries` topic руу хайлтын үйл явдал илгээнэ
6. **RAG боловсруулалт**: Knowledge Agent үйл явдлыг уншиж, vector хайлт хийж, үр дүнг `knowledge-results` topic руу бичнэ
7. **Service дуудлага**: Төлөвлөгөөний дагуу `service-calls` topic руу үйл явдлууд илгээгдэнэ
8. **Параллель гүйцэтгэл**: Service Agent-ууд параллель байдлаар үйл явдлуудыг уншиж, микросервисүүдийг дуудаж, үр дүнг `service-results` topic руу бичнэ
9. **Aggregation**: Response Aggregator бүх үр дүнг цуглуулж нэгтгэнэ
10. **Хариулт publish**: Эцсийн хариултыг `user-responses` topic руу бичнэ
11. **Хэрэглэгчид хүргэх**: API Gateway хариултыг уншиж хэрэглэгчид буцаана
12. **Мониторинг**: Бүх агент `monitoring-events` topic руу лог мэдээлэл бичнэ

Асинхрон онцлог: Агентууд бие биенийхээ хүлээхгүй. Тэд үйл явдлыг publish хийж, өөрийн ажилдаа үргэлжлүүлнэ. Энэ нь системийг өргөжүүлэх боломжтой, найдвартай болгодог.

4.2 Prompt engineering

4.2.1 Prompt загвар

Orchestrator Agent-ийн system prompt:

```
                    .        :

1.

2.

3.


           :

{service_list}


        :

{business_rules}
```

4.2.2 Service Registry

Сервис бүрийг JSON форматаар тодорхойлно:

```
{
  "name": "user-service",
  "description": "      ␣    ␣    ␣    ",
  "endpoints": [
    {
      "path": "/users/{id}",
      "method": "GET",
      "description": "      ␣    ␣  ",
```

```
      "parameters": {
        "id": "       ␣ID"
      }
    }
  ]
}
```

Код 4.1: Service Registry Example

4.2.3 RAG Систем

Knowledge Agent нь vector database ашиглана:

- **Index**: API баримт бичиг, өмнөх хүсэлтүүд, бизнес дүрэм
- **Embedding Model**: Sentence-Transformers эсвэл OpenAI embeddings
- **Vector DB**: FAISS, Pinecone, эсвэл Qdrant
- **Retrieval**: Асуулт бүрт top-k хамгийн холбогдолтой баримтуудыг олох

4.2.4 Алдаа

Алдаа гарах үед агент:

1. Алдааны төрөл, шалтгааныг тодорхойлно
2. Retry стратеги сонгоно (exponential backoff)
3. Альтернатив сервис байгаа эсэхийг шалгана
4. Хэрэглэгчид ойлгомжтой алдааны мэдэгдэл өгнө
5. Алдааны логийг хадгална

4.3 Моделийн давуу тал

4.3.1 Хиймэл оюун агентын онцлогууд

1. **Натурал хэл интерфейс**: Хэрэглэгч натурал хэл дээр хүсэлт илгээж болно
2. **Уян хатан орчуулалт**: Бизнес дүрэм өөрчлөгдөхөд код өөрчлөх шаардлагагүй
3. **Ухаалаг алдаа засалт**: Агент өөрөө алдааг таньж, засах арга санал болгоно
4. **Түүх санах**: Өмнөх харилцааг санаж, контекст хадгална
5. **Өгөгдлийн нэгтгэл**: Олон сервисийн өгөгдлийг нэгтгэж ойлгомжтой болгоно
6. **Өөрөө сайжрах**: Логуудаас суралцаж, цаашид илүү сайн ажиллана

4.3.2 Event-Driven давуу тал

1. **Салангид байдал (Decoupling)**:
    - Агентууд хоорондоо шууд хамааралгүй
    - Нэг агент унах нь бусдад шууд нөлөөлөхгүй
    - Агентыг өөрчлөх, солих хялбар
2. **Өргөжүүлэх чадвар (Scalability)**:
    - Агент бүр бие даан өргөжиж болно
    - Kafka partition ашиглан параллель боловсруулалт
    - Consumer group-оор ачааллыг хуваарилах
3. **Найдвартай байдал (Resilience)**:
    - Kafka-ийн replication алдагдал хамгаална
    - Үйл явдал дахин тоглуулах боломжтой
    - Агент тасалдах ч үйл явдал алдагдахгүй

4. **Бодит цагийн хариу үйлдэл**:
    - Үйл явдал тэр даруйдаа дамждаг
    - Streaming боловсруулалт
    - Бага хоцрогдол
5. **Олон хэрэглэгч дэмжлэг**:
    - Нэг үйл явдлыг олон систем ашиглаж болно
    - CRM, CDP, analytics руу автоматаар урсах
    - Шинэ consumer нэмэхэд producer өөрчлөх шаардлагагүй
6. **Observability ба Debugging**:
    - Бүх үйл явдал хадгалагдана
    - Lineage tracking: үйл явдал хаашаа явсныг мэдэх
    - Үйл явдал дахин тоглуулж bug олох
    - Дахин сургалтад ашиглах

4.3.3 Үр дүн

Энэхүү судалгааны ажил нь хиймэл оюун агентуудыг микросервис архитектурт нэвтрүүлж болох боломжит арга замыг харуулсан. Гол үр дүнгүүд:

1. Хиймэл оюун агентууд нь микросервис хоорондын нарийн төвөгтэй логикийг удирдаж чадна
2. RAG систем нь олон сервисийн мэдээллийг нэгтгэхэд үр дүнтэй
3. Натурал хэл интерфейс нь хэрэглэгчийн туршлагыг сайжруулна
4. Агент нь алдаа засалтын чадвартай
5. Систем нь уян хатан, өргөжүүлэх боломжтой

4.3.4 Хязгаарлалт

Санал болгосон загвар нь дараах хязгаарлалттай:

1. **Хоцрогдол**: LLM inference хоцрогдолтой. Бодит цагийн шаардлагатай системд тохиромжгүй.
2. **Өртөг**: API дуудлага бүр өртөгтэй. Өндөр ачаалалтай системд үнэтэй.
3. **Найдвартай байдал**: LLM-ийн найдвартай байдал 100% биш. Чухал системд баталгаат тохиргоо шаардлагатай.
4. **Аюулгүй байдал**: Prompt injection довтолгооноос хамгаалах шаардлагатай.

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] Devlin, J., et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT*, 2019.

[5] Brown, T., et al. "Language Models are Few-Shot Learners". *Advances in Neural Information Processing Systems*, 2020.

[6] Touvron, H., et al. "Llama 2: Open Foundation and Fine-Tuned Chat Models". *arXiv preprint arXiv:2307.09288*, 2023.

[7] Touvron, H., et al. "The Llama 3 Herd of Models". *arXiv preprint arXiv:2407.21783*, 2024.

[8] Lewis, P., et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". *Advances in Neural Information Processing Systems*, 2020.

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

[10] Wei, J., et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". *Advances in Neural Information Processing Systems*, 2022.

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

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

[13] Ortega, P.A., et al. "Shaping Representations Through Communication: Community Size and Stability Determine the Emergence of Shared Symbols". *DeepMind Technical Report*, 2021.

[14] Schulman, J. "Reinforcement Learning from Human Feedback: Progress and Challenges". *Berkeley EECS Colloquium*, 2022.

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

[16] Liu, P., et al. ”Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing”. *ACM Computing Surveys*, 2023.

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

[18] Richardson, Chris. *Microservices Patterns: With Examples in Java*. Manning Publications, 2018.

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

[20] Johnson, J., Douze, M., and Jégou, H. ”Billion-scale Similarity Search with GPUs”. *IEEE Transactions on Big Data*, 2019.

[21] Malkov, Y.A. and Yashunin, D.A. ”Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[22] Reimers, N. and Gurevych, I. ”Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. *EMNLP-IJCNLP*, 2019.

[23] Muennighoff, N., et al. ”MTEB: Massive Text Embedding Benchmark”. *EACL*, 2023.

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

[25] Python Software Foundation. ”Python 3 Documentation”. `https://docs.python.org/3/`

[26] Ramírez, Sebastián. ”FastAPI Documentation”. `https://fastapi.tiangolo.com/`

[27] OpenAI. ”OpenAI API Reference”. `https://platform.openai.com/docs/api-reference`

[28] Anthropic. ”Claude 3 Model Card”, 2024. `https://www.anthropic.com/claude`

[29] LangChain. ”LangChain Documentation”. `https://python.langchain.com/docs/`

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

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

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

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

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

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

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

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

[38] Wooldridge, Michael. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2009.

[39] Park, Joon Sung, et al. ”Generative Agents: Interactive Simulacra of Human Behavior”. *arXiv preprint arXiv:2304.03442*, 2023.

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

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

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