



Debrief — deepseek/deepseek-v3.2

- Instanser: 9
 - Tokens inn: 20,339
 - Tokens ut: 7,924
 - Kostnad: \$0.1799
 - Kjørt: 2026-04-14 16:11:26.295653+00:00
-

Sverm-debrief

Konsensus

1. **Arkitekturvalg:** Alle instanser anbefaler **Azure OpenAI + Python-baserte agenter** (kjørt i Azure Functions, Container Instances eller Container Apps) som den riktige veien for å bevare Microsoft-investering, sikkerhet og skalerbarhet. **Copilot Studio forkastes** for swarm-orkestrering, da den er for stiv og brukersentrert. **Proprietære rammeverk (CrewAI) frarådes** på grunn av risiko for skygge-IT, vendor lock-in og kompleksitet i et strengt helsedatamiljø.
2. **Data-sikkerhet og injeksjon:** Enstemmig enighet om at rå helsedata **aldri** må kopieres til eksterne API-er. Løsningen er **Azure AI Search med sikker filtrering** (via Entra ID) for å “grounde” agentene, eller direkte API-kall til SharePoint/Dataverse via Graph API. Data forblir i din Azure-tenant.
3. **Agent-spesialisering (Case 1):** Ikke lag 100 dupliserte agenter. Start med et begrenset sett (5-12) **distinkte kjerne-roller** (f.eks. regulatorisk, klinisk, økonomisk), hver med unik systemprompt og tilgang til spesifikke datakilder. Skalering til ~100 perspektiver oppnås via kontekst-filtre eller under-spørsmål generert av kjernagentene.
4. **Orkestrering og observabilitet:** **Azure Service Bus** eller **Queue Storage** anbefales som “inbox” for Case 2. **Azure Monitor/Application Insights** er kritisk fra dag én for å spore token-forbruk, latency og feilrater per agent, samt for kostnadskontroll.



Dissens

1. **Orkestreringsverktøy:** Det var delte meninger om det beste verktøyet for å koordinere agentene.
 - **En fraksjon** (SI_001, SI_006) fremhevet **Azure Durable Functions** for sin innebygde tilstandshåndtering og evne til å modellere komplekse, tilstandsfulle arbeidsflyter.
 - **En annen** (SI_003, SI_007) favoriserte **Azure Logic Apps** for sin lav-kode-tilnærming og enklere integrasjon med andre Azure-tjenester.
 - **En tredje** (SI_004, SI_005, SI_009) så på **Azure Container Instances/ Apps** eller **Azure Functions** som den mest fleksible og kontrollerbare plattformen for Python-basert agentlogikk.
2. **Aggregeringsmetode for Case 1:** Hvordan man best sammenstiller 100 perspektiver uten kaos.
 - Noen (SI_002, SI_003, SI_007) foreslo en dedikert **“syntese-” eller “meta-analyste-agent”** som manuelt eller semi-automatisk trekker ut konsensus.
 - Andre (SI_004, SI_005, SI_008) antydte mer avanserte metoder som **clustering-algoritmer** eller **vektet summarisering** for å identifisere både enighet og uenighet systematisk.
3. **Tilnærming til systemprompts for Case 2:** Om alle agenter skal dele én mal.
 - Flere (SI_004, SI_005, SI_007) argumenterte for en **felles core-template** med dynamisk parameterfylling for konsistens.
 - Motparten (SI_002, SI_006) foreslo **agent-pools med forskjellige maler** basert på case-kompleksitet for bedre kvalitet.

Blindsoner avdekket

Svermen avdekket at den største risikoen **ikke** er teknisk, men **prosessmessig og juridisk**:

- **Juridisk godkjenningssprosess:** En enkelt analyse kunne oversett behovet for en formell, juridisk og sikkerhetsmessig gjennomgang av selve dataflytsarkitekturen med IT-governance og juridiske avdelinger **før** noen PoC starter. Dette er avgjørende for helsedata.
- **Kill-switches og isolasjon:** Kompleksiteten ved å stoppe en “hallusinerende” agent som risikerer å korrumpere et helt svar eller lekket data, krever design av solide mekanismer for agent-isolasjon og manuell overstyring fra starten.



- **Definerte knekkpunkter:** Mangelen på forhåndsdefinerte, kvantitative grenser for kostnad og latency som automatisk stopper en kjøring og eskalerer til en menneskelig operator.

Anbefalinger

1. **Start med en juridisk/security-review av arkitekturen:** Få godkjenning fra IT-sikkerhet og juridisk avdeling for prinsippet om data-grounding via Azure AI Search/Graph API og bruk av Azure OpenAI i din region. Dette er **trinn null**.
2. **Bygg en minimal PoC med 3-5 spesialiserte agenter:** Fokuser på **Case 1**. Bruk Azure Durable Functions eller Python i Container Apps som orkestrator. Koble til en begrenset SharePoint-mappe via Azure AI Search. Mål: valider kontekst-injeksjon, mål kostnad/token-forbruk, og test en enkel syntesemetode (f.eks. en ekstra "sammendrags"-agent).
3. **Implementer omfattende observabilitet og knekkpunkter fra dag én:** Konfigurer Azure Monitor med dashboards for token-forbruk, feilrate og latency **per agent-type**. Definer klare terskler for når en kjøring skal stoppes automatisk. Dette bygger tillit og kontroll.
4. **Evaluer orkestrering for "inbox" (Case 2) sekundært:** Når PoC for Case 1 er stabil, utforsk en enkel kø-basert dispatcher med Azure Service Bus og én agent-type for en veldefinert micro-case (f.eks. klassifisering av henvendelser). Bruk felles prompt-mal med parametre.
5. **Kvantifiser verdi kontra enkelt-Copilot:** Gjennomfør en parallell analyse av et reelt problem med både standard Copilot og din 3-5 agents sverm. Mål og presenter forskjellene i analysedybde, dekning av perspektiver og tidsforbruk for å demonstrer klar ROI.