Protokol MCP - Kako povezati obstoječe aplikacije in agente

Jani Šumak

Inova IT d.o.o., Maribor, Slovenija
jani.sumak@inova.si

Novembra 2024 je družba *Anthropic*, najbolje poznana po razvoju skupine velikih jezikovnih modelov (angl. Large Language Models, v nadaljevanju LLM) po imenu Claude, objavila odprti protokol za modelni kontekst (angl. Model Context Protocol, v nadaljevanju protokol MCP). K sodelovanju so kmalu pristopili tudi drugi razvijalci in ponudniki rešitev, ki se poslužujejo velikih jezikovnih modelov, med drugim Google in OpenAI. Temeljni namen protokola je standardizacija interakcije med LLM aplikacijami (angl. LLM application) ter zunanjimi podatkovnimi viri in orodji. Protokol se osredotoča na vprašanje, kako velikim jezikovnim modelom in LLM aplikacijam podati kakovosten kontekst s pomočjo katerega lahko generirajo boljše rezultate ali izvajanje aplikacije (agenti). V uvodu bomo povzeli delovanje velikih jezikovnih modelov in izpostavili nekatere pomanjkljivosti. Na to bomo pregledali rešitve, ki naslavljajo enake ali podobne težave kot protokol MCP. Po uvodnem delu se bomo posvetili protokolu MCP. Prispevek bomo zaključili s prikazom konkretni implementaciji protokola MCP in primerov uporabe.

Ključne besede:
veliki jezikovni modeli,
pozivanje,
LLM Aplikacije,
protokol mcp,
agenti.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.17

1 Uvod

Z javno dostopnostjo prve iteracije aplikacije ChatGPT[22] in popularizacijo velikih jezikovnih modelov – tukaj gre zasluga tudi odprtokodnim modelom, kot so Llama[1] - so se pričela prizadevanja omejevanja stohastičnosti[2] aplikacij, ki uporabljajo velike jezikovne modele. Onstran pogovornih aplikacij (angl. chat-based applications) je (bila) integracija velikih jezikovnih modelov z obstoječimi ali novimi aplikacijami zahteven postopek.

Druga omejitev je meja znanja (angl. knowledge cutoff). Po končanem razvoju je model omejen na podatke, ki so bili na voljo tekom učenja in prilagajanja modela. V kolikor želimo, da ima model "dostop" do novejših podatkov ali podatkov, ki jih ni bilo v učnem naboru, na primer posodobljeno dokumentacijo ogrodja, ki ga uporabljamo, je potrebno omogočite vnos teh podatkov v poziv ali dovoliti, da model, natančneje aplikacija, ki uporablja model, te podatke pridobi in jih vključi v poziv.

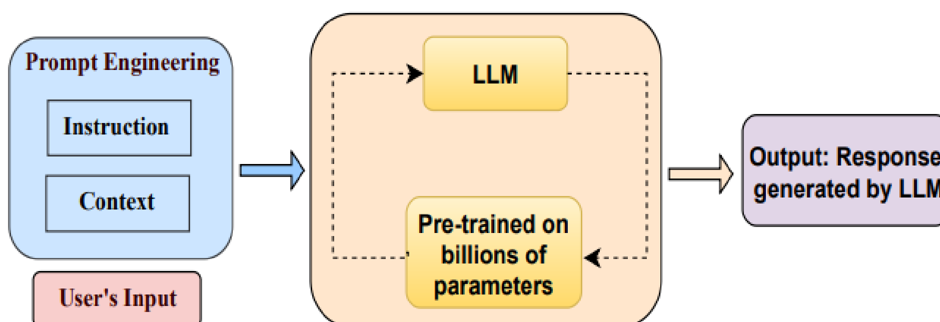
Rešitve, ki naslavljajo te omejitve, lahko razdelimo na tri pristope: nastavljanje dekodiranja[23], prilagajanje modelov in inženiring pozivov (ang. prompt engineering). Temeljna razlika med temi pristopi je, da se prva dva osredotočita na delovanje modela, medtem ko se slednji osredotoča vnos, natančneje na omejevanje procesa iskanja v vektorskem prostoru. Kot bomo pokazali, protokol MCP sodi ali dopolnjuje slednje rešitve.

2 Razvoj inženiranja pozivov

2.1. Pozivanje brez in z več primeri

Inženiring pozivov je veščina oblikovanja pozivov, ki poveča zmogljivosti velikih jezikovnih in vizualno-jezikovnih modelov. Poziv je sestavljen iz navodil, konteksta in uporabnikovega vnosa. [3] Pristop je precej razširjen in se ga poslužujejo tudi večji ponudniki velikih jezikovnih modelov, na primer v obliki sistemskih pozivov.[4]

Povezave pozivov in delovanja modela so opazili raziskovalci prvih modelov GPT. Namesto specializiranih modelov za posamezna področja, so raziskovalci ugotovili, da lahko uporabijo namensko pripravljene pozive s katerimi so lahko model uspešno uporabili na nalogah za katere ni imel posebej pripravljenih podatkov. [5] Tej tehniki, pozivanje brez primerov (angl. zero-shot prompting) je sledil poizkus, da v pozivu navedemo več primerov, t.i. pozivanje z več primeri (angl. few-shot prompting, ki še dodatno optimizira rezultat.[6]



Slika 1: Inženiranje pozivov. [7]

[22] ChatGPT je bil javno dostopen 30. novembra 2022. Prva iteracije je uporabljala model GPT 3.5.

[23] Gre za spreminjanje nastavitev modela, ki vplivajo na ustvarjeni rezultat. Uporabljajo se parametri, kot sta temperatura in top-p.

Oba pristopa ste v osnovi enaka. Pozivanje brez primerov pred uporabnikov poziv doda dodatna navodila in po potrebi kontekst. Pozivanje z več primeri pred uporabnikov poziv doda enega ali več primerov rešitve naloge. Čeprav oba pristopa dosežeta boljše rezultate z istimi modeli, je potrebno veliko truda za pripravo kakovostnih pozivov.

2.2. Verižno sklepanje

Kljub uspešnosti zgoraj opisanih pristopov veliki jezikovni modeli niso uspešno reševali naloge, kjer je bilo potrebno simulirati razmišljanje, na primer artimetika. Jason Wei je s soavtorji predstavil nov pristop, t.i. verižno sklepanje (angl. chain of thought). Verižno sklepanje model vodi v dodajanje t.i. vmesnih miselnih korakov. Namesto, da bi model generiral rezultat, model generira vmesne korake, ki vodijo do natančnejšega rezultata. [8]

Podobno kot pri pozivanju z več primeri je priprava kakovostnih pozivov za verižno sklepanje dolgotrajen in zahteven postopek. Kmalu so raziskovalci odkrili in preizkusili tehnike, ki do neke mere avtomatizirajo postopek verižnega sklepanja. Med bolj znanimi je tehnika, kjer se pozivu doda preprost stavek, da naj postopa po korakih, npr. "Let's think step by step."[9]

2.3. Uporaba orodji

Shunyu Yao je s sovatorji združil ugotovitve verižnega sklepanja in raziskovanja zmožnosti velikih jezikovnih modelov, da načrtujejo, in predlagali uporabo orodji. Verižno sklepanje in sorodne tehnike ne odpravijo omejitve na učne podatke modela. Kolikor se modelu doda možnost vključevanja zunanjih virov, na primer dostopa do programskega vmesnika Wikipedija, se bistveno izboljšajo generiranje rezultatov. [10]

Zmožnost modelov, da kličejo funkcije ali uporabljajo zunanja orodja, se sicer lahko doseže na način kot je opisan v zgoraj navedenem članku, a postopek težko skaliramo. Modele se lahko nauči uporabe orodja nadzorovanim učenjem, kar poenostavi integracijo novih orodji. [11]

Na temelju tovrstnih ugotovitev je družba OpenAI je modelom GTP 3.5 in GTP 4 dodala zmožnost klicanju funkcij in uporabe orodji. [12] Omenjena modela sta lahko uporabljala v naprej pripravljene funkcije ali funkcije, ki jih je pripravil uporabnik, tako da se je klicu dodal seznam orodji. Model pri generiranju rezultata odloči ali uporabo eno ali več funkcij ali ne. V času nastajanja tega prispevka je večina funkcij že dodanih v grafični vmesnik orodja ChatGPT.

2.4. Dodatek: agenti

Preden se posvetimo protokolu MCP, ki neposredno odgovarja na zastavljeno težavo, je potrebno omeniti koncept agentov. Agenti sicer niso neposredno povezani s protokolom MCP ali razvojem pozivnega inženiringa, se pa pogostop pojavljajo v povezavi z uporabo orodji in tehnikami priprave pozivov.

Agenti so aplikacije, kjer generiran rezultat velikih jezikovnih modelov nadzoruje potek izvedbe. Kadar govorimo o agentih ne govorimo o samostojnih aplikacijah, ki sami nadzorujejo svojo izvedbo, ampak o zmožnosti aplikacije, da v potek izvedbe vključi generiran rezultat modela. Agenti imajo lahko večje ali manjše stopnje integracije in uporabe rezultatov modela, zato govorimo o spektru agentskih aplikacij. Kar te aplikacije povezuje je omenjen vpliv generiranega rezultat na izvedbo. [13]

Agenti so seveda možni brez klicanja orodji in ne potrebujejo naprednih tehnik pozivov. Toda ker agenti delujejo z zunanjimi viri, na primer posodobljajo dokument na našem računalniku, si je v praksi težko zamisliti agente, ki ne lahko uporabljajo orodja in se ne poslužujejo tehniki pozivanja kot smo jih omenili zgoraj.

3 Protokol MCP

3.1. O protokolu

Če se vrnemo k uporabi funkcij in orodji, smo nakazali, da je težava število integracij, ki jih je potrebno narediti, da lahko integrirano programski vmesnik z modeli ali ponudniki modelov.

Anthropic je dne 25. Novembra 2024 objavil prvo revizijo protokola MCP. [14] Protokol je naslovil težavo številnih integracij, ki bi jih morali izdelati, da bi asistenti – njihov izraz za aplikacije, ki uporabljajo velike jezikovne modele – lahko uporabljali zunanje vire ali obstoječe aplikacij.

Nedolgo za tem sta protokol podprla tudi OpenAI in Google. Prvi implicitno z vključitvijo v njihovo ogrodje za razvoj agentov [15], slednji z izrecno omembo na konferenci Google I/O. S podporo treh največjih ponudnikov velikih jezikovnih modelov je protkol postal *de facto* standard.

Protokol nastaja javno, na omrežju Github. [16] Vsakdo lahko prispeva ali začne razpravo o protokolu na omrežju Github. V času nastajanja tega članka je zadnja različica protokola 3. različica z dne 18. junija 2025. Vsaka nova različica v uvodnem delu vključuje seznam sprememb v primerjavi s prejšnjo različico.

3.2. Osnove protokola

Protokol MCP je razdeljen na naslednja poglavja: uvod, osnovni protokol, odjemalec, strežnik in shema. Protokol standardizira integracijo ne glede na končno uporabo aplikacije, bodisi da gre za integrirano okolje za razvijalce, pogovori vmesnik ali po meri narejeno rešitev z uporabo LLM-ov. Protokol določa tako arhitekturni vidik, kot tudi format komunikacije.

Kot bomo videli v nadaljevanju protokol uporabi protokol JSON-RPC 2.0[17] za pošiljanje sporočil med strežnikom in odjemalcem. Kot analogijo protokol izrecno omenja protokol za jezikovne strežnike.

MCP standardizira:

- Deljenje konteksta z velikimi jezikovnimi modeli,
- Deljenje informacij o orodjih in zmogljivostih stražnikov in zmogljivostih odjemalcev,
- Izgradnjo izmenljivih in združljivih integracij terdelovnih potekov.

Protokol uporablja *JSON-RPC 2.0* sporočila za vzpostavitev komunikacije med:

- **Gostitelji**: LLM aplikacije, ki vzpostavijo povezave,
- **Odjemalci**: sestavni deli gostiteljske aplikacije, ki skrbijo za povezavo s strežnikom,
- **Strežniki**: proces, ki zagotavljajo kontekst in zmogljivosti.

MCP predpostavlja stanovitno povezavo med odjemalcem in strežnikom, ki komunicirata s pomočjo protokola JSON-RPC.

Strežniki odjemalcem nudijo:

- Vire: kontekst in podatke,
- Pozive: predloge sporočil in delovnih potekov,
- Orodja: funkcije, ki jih lahko LLM aplikacija pokliče.

Dodatno lahko **odjemalec** strežnikom nudi:

- Vzorčenje: strežniki lahko zaženejo delovanje agenta ali rekurzivno interakcijo z velikim jezikovnim modelom,
- Korene: odjemalec lahko strežniku omeji poizvedbe o enoličnih identifikatorjih ali dotočenem sistemu,
- Pridobivanje: poizvedbe s strani strežnika za pridobivanje dodatnih informacij o uporabniku.

Varnost in zaupanje sta sestavna dela protokola. Protokol predpisuje in zahteva, da se vsa dejanja izvajajo na podlagi izrecne privolitve. Postopek privolitve mora biti jasen, zato morajo biti tudi vmesniki razumljivi. Podatki, ki jih gostitelj ali strežnik hranita, ne smejo biti hranjeni brez uporabnikovega privoljenja. Podatki morajo biti ustrezno zaščiteni.

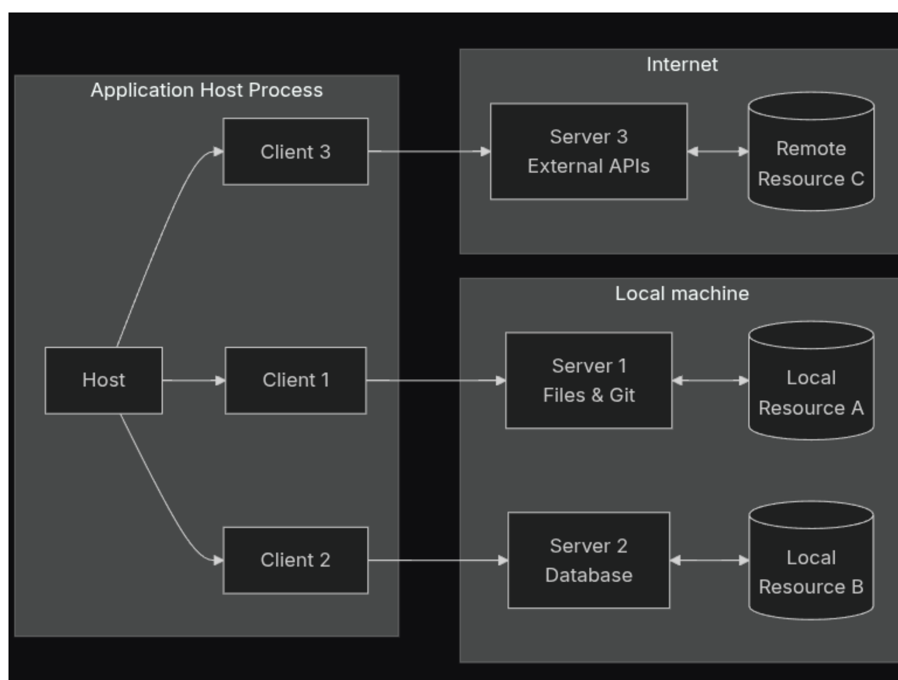
Nadalje protokol zahteva, da so orodja, ki jih nudi strežnik odjemalcu, varna in se smejo uporabiti samo s privoljenjem uporabnika. Njihova uporaba mora biti jasna in razumljiva uporabniku. Tudi za vzorčenje velja, da lahko strežniki uporabijo vzorčenje samo, če so meje vzorčenja jasne in je uporabnik predhodno privolil.

Protokol ne more vsiliti ali zagotoviti varnostih omejitev in zahtev, zato razvijalcem zadaja, da morajo graditi robustne postoka avtorizacije in podajanja soglasja, zagotavljati jasno dokumentacijo o varnosti sistema, implementirani ustrezen nadzor dostopanja in zaščite podatkov, slediti varnostnim napotkom in dobrim praksa ter vključiti varnost v vsako funkcionalnost njihovega sistema.

Polega navedenega protokol določa tudi načine nastavljanja, sledenja napredovanja, prekinitve izvajanja, sporočanja napak in beleženja.

3.3. Arhitektura

Kot je že bilo omenjeno so osnovne komponente protokola gostitelj, odjemalec in strežnik. Vsak gostitelj ima lahko več odjemalcev, tako da sta odjemalca in aplikacija ločena, kar omogoča, da se isti odjemalec doda več aplikacijam in ena aplikacija lahko vključuje različne odjemalce.



Slika 2: Inženiranje pozivov.[16]

Gostitelj:

- upravlja več odjemalcev,
- nadzoruje povezave, dovoljenja in življenjske cikle odjemalcev,
- uveljavlja varnost,
- razrešuje zahteve za avtorizacijo,
- usklajuje integracijo z velikim jezikovnimi modeli in vzorčenjem,
- upravlja agregacijo konteksta med odjemalci.

Odjemalec:
- za vsako sejo vzpostavi stanovitno povezavo s strežnikom,
- izvaja pogajanje o protokolu in izmenjavo zmogljivosti,
- dvosmerno usmerja protokolna sporočila,
- uprava z naročanjem in obvestili,
- zagotavlja varnost med strežniki.

Strežniki:
- Izpostavijo vire, orodja in pozive z uporabo osnovnih gradnikov protokola,
- Delujejo neodvisno od odjemalca,
- Zahteve vzorce,
- Spoštuje varnostne omejitve,
- Lahko deluje lokalno ali kot storitev na daljavo,
- Naj bo enostaven in se omeji na jasno določene zmogljivosti, kompleksne naloge naj prepusti gostitelju,
- Naj deluje izolirano,
- Ne smejo dostopati do celotnega pogovora ali imeti vpogleda v druge strežnike - za komunikacijo med strežniki skrbi gostitelj,
- Strežniki in odjemalci lahko postopoma dodaja funkcionalnosti in se razvijajo neodvisno drug od drugega.

Pogajanje o zmogljivostih

Kot je razvidno protokol določa majhen nabor zmogljivosti odjemalca in strežnika. Da bi odjemalec in strežnik vedela kater funkcionalnosti imata na voljo, morat sporočiti zmogljivosti in njihove omejitve.

Ob inicializaciji strežnik odjemalcu sporoči nabor naročnin, orodji in predlog za pozive. Odjemalec sporoči strežniku kater zmogljivosti podpira, npr. vzorčenje, obvestila. Oba morata spoštovati zmogljivosti in omejitve tekom seje.

3.4. Transportni protokoli, življenjski cikel, avtorizacija, varnost in pripomočki

MCP določa dva standardna transporta:

- **stdio**: strežnik se izvaja kot lokalni proces in komunicira prek stdin in stdout.

- **pretočni HTTP** (angl. streamable HTTP): strežnik uporablja protkol HTTP, strežnik palahko odgovori z enim samim JSON objektom ali s tokom SSE.

Življenjski cikel sledi točno določenim korakom:

- inicializacija seje: prvotna zahteva, prvoten odziv in potrditev,
- faza uporabe,
- postopek izklopa,
- prekinitev povezave.

V postopku inicializacije odjemalec strežniku pošlje različico protokola, podatke o sebi in svojih zmogljivostih. Strežnik mora potrditi različico protokola in poslati lastne podatke in seznam zmogljivosti.

V fazi uporabe odjemalec in strežnik spoštujeta različico protokola, ki sta jo dogovorila, medtem ko se nabor zmogljivosti strežnika lahko spreminja, tako da strežnik pošlje odjemalcu sporočilo, da naj odjemalec posodobi seznam zmogljivosti.

V postopku izklopa protokol predvideva, da se uporabijo metode transportnega mehanizma.

Protokol MCP postavlja varnost v ospredje z implementacijo OAuth 2.1.[18] Strežniki delujejo morajo preverjati žetone. Žetoni se pridobijo za vsak strežnik posebej in se ne smejo deliti.

Nadalje protokol nalaga razvijalcem, da:

- validirajo vhodne podatke in žetone,
- nadzorujejo dovoljenja in dostope ter preprečujejo pisanje zaupnih podatkov v sporočila ali dnevnike,
- seje naj bodo varne in edinstvene.

3.5. Koreni, vzorčenje in pridobivanje

Odjemalca komunicirajo s strežnikom in strežnik postavljajo meje ali kontekst. Odjemalci lahko dajo strežniku na voljo dostop do lokalnega datotečnega sistem, a ga lahko sočasno omejijo na enega ali več korenov.

Odjemalec v fazi inicializacije navede zmogljivosti, ki jih podpirajo. Podroben seznam pa pošljejo na strežnikovo poizvedbo z ustrezno metodo.

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "roots/list"
}
```

Slika 3: Zahteva za seznam korenskih dostopov.

Vzorčenje je zmožnost odjemalca, da strežniku omogoči dostop do modela. Strežnik model uporabi, da oblikuje sporočila, generira slike ipd. Na ta način lahko tudi strežnik deluje kot agent in za izvajanje ukaza uporabi model, ki ga uporablja aplikacija. Odjemalec lahko strežniku izpostavi dodane informacije o modelih, ki so na voljo, kot na primer hitrost, ali celo namigne, kako naj modele izbira.

Kadar strežnik za izvedbo ukaza potrebuje dodatne informacije s strani uporabnika lahko pridobi dodatne informacije od odjemalca, v kolikor jih odjemalec podpira. Postopek poteka podobno kot izvajanje ukazov, pri

čemer je strežnik tisti, ki sledi navodilo za izvajanje ukaza ali poizvedbe in odjemalec validira in odobri ali zavrne poizvedbo.

3.6. Pozivi, viri, orodja in pripomočki

Trije primitivni tipi MCP protokola so pozivi, viri in orodja. Pozivi so predloge, ki jih uporablja in nadzira uporabnik, na primer s poševnico v pogovorni aplikaciji. Viri so datoteke in drugi besedilni viri, ki jih pridobi in uporablja aplikacija. Orodja so funkcije, ki jih lahko uporabi LLM s klici na MCP strežnik.

Strežnik mora v fazi inicializacije sporočiti kater primitivne tipe podpira.


```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "capabilities": {
      "prompts": {},
      "resources": {},
      "tools": {}
    },
    "protocolVersion": "2025-06-18",
    "serverInfo": {
      "name": "magnum-mcp-server",
      "version": "1.0.0"
    }
  }
}
```


Slika 4: Inicializacija strežnika.

Strežnik ne rabi navesti vseh primitivov, ki jih uporablja, mora pa jih navesti. Seznam se pridobi s pošiljanjem ukaza s pravilno navedeno metodo, npr. Prompts/list.


```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "prompts": [
      {
        "arguments": [
          {
            "description": "Name of the new workspace",
            "name": "workspace_name",
            "required": true
          },
          {
            "description": "Email of the workspace administrator",
            "name": "admin_email",
            "required": true
          }
```


Slika 5: Seznam pozivov.

Vsak primitivni tip ima drugačno podatkovno strukturo. Pozivi, kot je razvidno, imajo ime, naziv (opcijsko), opis (opcijsko) in seznam argumentov. Od odjemalca pa pričakujejo vlogo (uporabnik ali asistent) in vsebino. Vsebina je nadaljnje razdelana glede na tip, torej besedilo, zunanji vir, slike ipd.

Podobnemu vzorcu sledijo viri in orodja. Strežnik navede seznam virov ali orodji v predpisani podatkovni strukturi, odjemalec pa pošlje zahtevo v predpisani obliki. Viri so morda posebnost, ker ne pričakujejo vsebine, ampak pravilno oblikovan enolični identifikator vira. Pri oblikovanju enoličnih identifikatorjev protokol MCP ne omejuje nabora, navaja pa splošne sheme kot so https://, file:// in git://.

Poleg primitivnih tipov strežniki lahko omogočajo tudi zmožnosti dopolnjevanja za izboljšano interaktivnost, beleženj in paginacijo.

3.7. Shema

Protokol vključuje tudi shemi, kjer so definirani ukazi, podatkovni modeli in tipi. Za shemi uporablja programski jezik Typescript[19] ali JSON shemo[20], ki se ga s pomočjo orodji enostavno pretvori v druge programske jezike, kot na primer vmesnike (angl. interface) v jeziku golang.

Shema

4 Praktični primeri

Na omrežju Github je v okviru organizacije Model Context Protocol objavljen seznam vzorčnih in vidnejših strežnikov.[21] Prav tako je na zgoraj navedeni spletni strani prosto dostopni tečaj, kjer je prikazan postopek implementacije strežnika.

Kljub zapletenosti protokola, predvsem na mestih, ki omogoča agentske lastnosti na strežniku, je implementacija strežnika dokaj enostavna. Na omrežju Github je na voljo več paketov za razvoj programske opreme za različne programske jezike. [22]

4.1. Strežnik za dostop do datotečnega sistema

Če pogledamo enega izmed uradnih vzročnih primerov, strežnik za dostop do datotečnega sistema. Strežnik je vzorčna implementacija, zato koda ne sledi nujno vsem dobrim praksam.

Strežnik se nahaja v eni datoteki. Ko zažene strežnik navede, da podpira orodja.

```
// Server setup
const server = new Server(
  {
    name: "secure-filesystem-server",
    version: "0.2.0",
  },
  {
    capabilities: {
      tools: {},
    },
  },
);
```

Slika 6: Strežnik MCP za datotečni sistem – inicializacija strežnika.

Seznam orodji našteje v namenskem upravljalniku.

```
// Tool handlers
server.setRequestHandler(ListToolsRequestSchema, async () => {
  return {
    tools: [
      {
        name: "read_file",
        description: "Read the complete contents of a file as text. DEPRECATED: Use read_text_file instead.",
        inputSchema: zodToJsonSchema(ReadTextFileArgsSchema) as ToolInput,
      },
      {
```

Slika 7: Strežnik MCP za datotečni sistem – seznam orodji.

Odjemalec kliče orodja v skladu s protokolom, strežnik pa jih pokliče tako, da iz ukaza pridobi orodje in ga požene z namensko metodo.

```
server.setRequestHandler(CallToolRequestSchema, async (request) => {
  try {
    const { name, arguments: args } = request.params;

    switch (name) {
      case "read_file":
      case "read_text_file": {
        const parsed = ReadTextFileArgsSchema.safeParse(args);
        if (!parsed.success) {
          throw new Error(`Invalid arguments for read_text_file: ${parsed.error}`)
        }
        const validPath = await validatePath(parsed.data.path);
```

Slika 8: Strežnik MCP za datotečni sistem – uporaba orodji.

Strežnik lahko zažene lokalno z ukazom "*npx -y @modelcontextprotocol/server-filesystem ~*". Strežnik uporablja protokol transportni protokol stdio in povežete s svojo aplikacijo LLM.

4.2. Strežnik za dostop do internega sistema za beleženje projektov

Na družbi Inova IT imamo interno orodje za beleženje dela na projektih. Aplikacija dnevno poziva zaposlene, da sporočijo na katerem projektu so delali. Ker je delo na projektih povezano z rednimi in izrednimi odsotnostmi od dela, aplikacija beleži in omogoča upravljanje dopustov.

Za potrebo tega prispevka in interno demonstracijo smo implementirali vzorčni strežnik za dostop do zalednega sistema aplikacije. Strežnik ima na voljo vzorčne pozive, orodja in celo izvede prijavo. Pri slednjem sicer ne sledi vse varnostmi zahtevam protokola, saj gre za vzorčno implementacijo.

Strežnik je bil zgrajen pomočjo orodja Claude code. [23]. Agent je dobil dostop do specifikacije Open API 3.0, ki jo s pomočjo anotacij samodejno generira zaledni sistem. S pomočjo specifikacije programskega vmesnika je agent pripravil ostnutek strežnika s seznamom pozivov in orodji. Tako kot pri zgornjem primeru so orodja del kode in za njih skrbi namenski upravljalnik.

```
func (s *MCPServer) handleRequest(ctx context.Context, req *MCPRequest) *MC
  response := &MCPResponse{
    Jsonrpc: "2.0",
    ID:      req.ID,
  }

  switch req.Method {
  case "initialize":
    response.Result = s.handleInitialize(req.Params)
  case "notifications/initialized":
    return nil
  case "tools/list":
    result, err := s.handleToolsList(req.Params)
    if err != nil {
      response.Error = &MCPError{
        Code:    ErrorInternal,
        Message: err.Error(),
      }
    } else {
      response.Result = result
    }
  case "tools/call":
    result, err := s.handleToolCall(ctx, req.Params)
    if err != nil {
      // Map error messages to appropriate error codes
      code := ErrorToolExecutionFailed
      if strings.Contains(err.Error(), "unknown tool") {
        code = ErrorToolNotFound
      } else if strings.Contains(err.Error(), "authentication required") {
        code = ErrorAuthenticationRequired
      } else if strings.Contains(err.Error(), "permission denied") {
```

Slika 9: Upravljalnik za orodja.

Strežnik smo uporabili v zgoraj omenjenim Claude code. Čeprav Claude code ni namenjen splošni uporabi, ampak je prilagojena za delo s kodo, je bil rezultat dober, saj je bilo mogoče podati zahtevo za izpis delovnih okolji in ustvariti novo okolje.

Primeru uporabe, ki so bili preizkušeni, so sicer preproste. Drži tudi, da je trenutno grafični vmesnik internega orodja boljši za izvedbo teh ukazov, a morebitne bodoče integracije s koledarjem, kjer bi zahtevo za dopust primerjali s stanjem v uporabnikovem koledarju nakazujejo na zanimive primere uporabe.

Strežnik je napisan v programskem jeziku Golang z minimalno uporabo knjižnic. Koda je dostopna na Githubu v organizaciji *inovait*.[24]

5 Zaključek

Avtor ocenjuje, da je protokol MCP velik korak v razvoju aplikacij, ki uporabljajo velike jezikovne modele. Protokol MCP sicer ne rešuje temeljnih omejitev velikih jezikovnih modelov, omogoča pa lažjo izgradnjo aplikacij, ki jih uporabljajo. Uporaba, kot je bilo nakazano, ni omejena na uporabnike, ampak je lahko del večjega procesa, ki ga deloma ali celoti izvajajo agenti.

Kljub številnim primerom implementacije in podporo programskim jezikom, je implantacija strežnika zahtevno delo. Tukaj avtor opozarja predvsem na varnostne zahteve. Aplikacije, ki del izvajanja prepustijo velikim jezikovnim modelom ali agentom, ki jih uporabljajo, izpostavljajo velike površine za napade. Agenti lahko pobrišejo datoteke ali baze, delijo zaupne ali osebne podatke ali prek predlog pozivi okužijo aplikacije.

Literatura

[1] huggingface.co/docs/transformers/en/model_doc/llama, Modeli Llama, obiskano 29. 7. 2025

[2] BENDER Emily M., GEBRU Timnit, McMILLAN-MAJOR Angelina, SHMITCHELL Shmargaret "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?", Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, Association for Computer Machinery – ACM, New York, str. 616.

[3] SAHOO Pranab, AYUSH Kumar Singh, SRIPARNA Saha, VINIJA Jain, SAMAT Sohel Mondal, AMAN Chadha "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications." ArXiv abs/2402.07927 (2024), str. 1.

[4] NEUMANN Anna, KIRSTEN Elisabeth, ZAFAR Muhammad Bilal, SINGH Jatinder "Position is Power: System Prompts as a Mechanism of Bias in Large Language Models (LLMs)", Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency, Association for Computing Machinery, New York, str. 574.

[5] RADFORD Alec, WU Jeffrey, CHILD Rewon, LUAN David, AMODEJ Dario, SUTSKEVER Ilya "Language models are unsupervised multitask learners". OpenAI blog, 1(8):9, 2019, str. 2.

[6] BROWN Tom et al "Language models are few-shot learners", arXiv preprint arXiv:2005.14165, str. 6.

[7] SAHOO Pranab, str. 1.

[8] WEI Jaso et al. "Chain-of-thought prompting elicits reasoning in large language models", Advances in Neural Information Processing Systems, številka 35, letnik 24824–24837, 2022, str. 3.

[9] SAHOO Pranab, str. 2.

[10] YAO Shunyu, ZHAO Jeffrey, YU Dian, DU Nan, SHAFRAN, Izhak, NARASIMHAN Karthik, CAO, Yuan "ReAct: Synergizing Reasoning and Acting in Language Models", International Conference on Learning Representations, ICLR 2023, Ruanda, str. 3-4.

[11] YUIJA Qin et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs, arXiv preprint arXiv:2307.16789, str. 3.

[12] https://platform.openai.com/docs/guides/tools, Navodila za uporabo orodji, obiskano dne 29. 7. 2025

[13] https://huggingface.co/docs/smolagents/en/conceptual_guides/intro_agents, Spletni tečaj za delo na agentih, obiskano dne 29. 7. 2025

[14] https://www.anthropic.com/news/model-context-protocol, Protokol MCP, obiskano dne 29. 7. 2025

[15] https://github.com/modelcontextprotocol/modelcontextprotocol, Repozitorij protokola MCP, obiskano dne 29. 7. 2025

[16] https://openai.github.io/openai-agents-python/mcp, obiskano dne 29. 7. 2025

[17] https://www.jsonrpc.org/, obiskano dne 29. 7. 2025

[18] https://modelcontextprotocol.io/specification/2025-06-18/architecture, obiskano dne 29. 7. 2025

[19] https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/ OAuth 2.1, obiskano dne 29. 7. 2025

[20] https://github.com/modelcontextprotocol/modelcontextprotocol/tree/main/schema/2025-06-18/schema.ts, , obiskano dne 29. 7. 2025

[21] https://github.com/modelcontextprotocol/modelcontextprotocol/tree/main/schema/2025-06-18/schema.json, , obiskano dne 29. 7. 2025

[22] https://github.com/modelcontextprotocol/servers, Seznam MCP strežnikov, obiskano dne 29. 7. 2025

[23] https://github.com/modelcontextprotocol?q=sdk&type=all&language=&sort=, Seznam paketov za razvoj programske opreme, obiskano dne 29. 7. 2025

[24] https://www.anthropic.com/claude-code, Claude code, obiskano dne 29. 7. 2025

[25] https://github.com/inovait/magnum-mcp, Repozitorij MCP strežnika za interno orodje na Inova IT, obiskano dne 7. 8. 2025.