Interogare LLM pentru instrumente de verificare formală

Rancov Larisa

Universitatea de Vest din Timișoara, Facultatea de Informatică
Master: Inginerie Software
Timișoara, România
larisa.rancov03@e-uvt.ro

Abstract. Acest raport prezintă dezvoltarea unei aplicații care folosește un model de limbaj de mari dimensiuni (LLM) pentru a identifica periodic instrumente din domeniul verificării formale și pentru a le organiza într-un catalog ușor de consultat. Scopul este reducerea efortului de descoperire a unor tool-uri relevante (corectitudine funcțională, analiza terminării, limite de complexitate, verificarea rețelelor neuronale și solvere QBF), într-un context în care informația este dispersată și se învechește rapid. Sistemul generează propuneri în format structurat (CSV), extrage metadate minimale (descriere, funcționalități, limbaje suportate, licență, versiune/actualizare, URL oficial) și le normalizează înainte de stocare. Pentru a limita riscurile asociate halucinațiilor LLM, propunerile nu sunt afișate automat, ci sunt salvate inițial în starea pending și devin vizibile doar după o verificare umană (approve/deny). Implementarea este realizată cu FastAPI pentru backend și React pentru frontend, iar în testare s-a folosit un model local prin Ollama. Rezultatul final este un flux end-to-end repetabil, care accelerează trierea inițială a instrumentelor și menține controlul asupra calității informațiilor afișate.

Keywords: instrumente de verificare formală · human-in-the-loop · LLM

1 Introducere

Verificarea formală a programelor a devenit tot mai prezentă în proiectarea software, având cerințe mari de fiabilitate, de la sisteme critice și infrastructură până la aplicații utilizate în mai multe domenii. În paralel, ecosistemul s-a diversificat rapid fiindcă au apărut instrumente specifice pentru diferite sarcini, precum dovedirea corectitudinii funcționale, demonstrarea terminării, estimarea limitelor de complexitate, verificarea proprietăților pentru rețele neuronale și evaluarea formulărilor booleene cuantificate. Ritmul aparițiilor, repoziționărilor și abandonărilor de proiecte face dificilă menținerea unei imagini actualizate a peisajului, informațiile fiind dispersate în repository-uri și rezultate ale competițiilor, iar un catalog întreținut manual își pierde actualitatea rapid.

Dificultatea reală nu provine doar din volum, dar și din calitatea neuniformă a metadatelor publice. Există descrieri sumare sau ambigue, licențe neprecizate ori schimbate între versiuni, categorii amestecate și semnale slabe privind activitatea curenta a proiectului care îngreunează selecția. Fenomene precum link-rot sau reorganizarea repository-urilor duc la linkuri invalide, iar deduplicarea devine dificilă cand același instrument este reambalat, redenumit sau multiplicat prin fork-uri. În absența unui proces sistematic, listele publice devin incomplete sau inconsecvente, ceea ce descurajează folosirea si reduce utilitatea de cercetare.

LLM-urile mari pot accelera descoperirea inițială, deoarece sunt utile pentru a formula și rafina interogări, a sumariza pagini, a extrage nume, descrieri, linkuri, indicii de licență și o încadrare preliminară pe categorie. Aceste avantaje vin însă cu riscuri precum halucinații, clasificari inexacte, confuzii între „bibliotecă" , „framework" , și „instrument utilizabil direct" , dar și cu tendința de a acorda aceeași greutate surselor cu credibilitate diferită. Din acest motiv, un flux complet automat nu ar fi potrivit pentru un catalog public care își propune rigoare și trasabilitate. Este necesară o etapă de control uman înainte de publicare, care să confirme existența proiectului, corectitudinea încadrării și validitatea informațiilor.

Proiectul de față propune o abordare echilibrată, ce include un flux de descoperire asistat de LLM, rulat periodic, completat cu validare umană. Sistemul interoghează LLM-ul pe direcții corespunzătoare categoriilor vizate (corectitudine funcțională, terminare, limite de complexitate), extrage metadate minimale, le normalizează (nume, descriere scurtă, adresă de repo sau pagină oficială, limbaj, categorie), verifică automat existența linkurilor. Propunerile trec apoi printr-o interfață de revizuire, în care editorul uman poate aproba sau respinge metadatele, fiindca doar intrările aprobate se propagă către lista finală.

Scopul proiectului este să identifice în mod recurent instrumente noi de verificare formală care merită atenție, punând accent pe noutate, relevanță și pe calitatea informației care le însoțește. Demersul este important pentru că reduce costul de căutare pentru cercetători și practicieni, scurtează timpul până la evaluări comparative și experimente, scoate la lumină proiecte recente sau nișate care altfel rămân greu de descoperit și oferă un punct de plecare suficient de clar pentru a decide dacă un instrument merită explorat mai departe. În acest sens, LLM-ul are rolul de a propune piste plauzibile, iar verificarea umană

funcționează ca filtru de încredere, astfel încât recomandările finale să fie utile și responsabile. Obiectivul este deci furnizarea constantă de sugestii proaspete si bine motivate, care să susțină activitatea de cercetare și integrare practică.

2 Descrierea problemei

Problema abordată este identificarea, la intervale regulate, a unor instrumente pentru verificarea formală a programelor și prezentarea lor, într-o formă coerentă utilizatorilor care decid dacă merită recomandate mai departe. Prin descoperire periodică se înțelege reluarea aceleiași proceduri de căutare și filtrare la intervale fixe, astfel încât rezultatele să nu depindă de un efort punctual, ci să producă în timp o serie constantă de propuneri. Human-in-the-loop face referire la faptul că decizia de includere nu este automată, ci un utilizator verifică existența proiectului, sensul încadrării pe categorie și caracterul plauzibil al informațiilor asociate, iar recomandarea finală se realizează doar după această confirmare.

Domeniul adoptat este cel al verificării formale, cu accent pe categoriile menționate mai sus. Interesul cade pe instrumente noi sau mai puţin vizibile, utile pentru a inspira experimente, comparaţii sau adoptări rapide în proiecte.

Pentru ca o propunere să fie evaluată eficient, fiecare instrument identificat trebuie să fie însoțit de un set minimal de informații funcționale, suficient pentru o înțelegere de primă lectură. Sunt necesare deci un nume clar și o scurtă descriere în câteva fraze, un link către repository sau către pagina oficială, o indicare a limbajului ori a tehnologiilor predominante, menționarea licenţei atunci când poate fi dedusă din sursa oficială, o etichetă de categorie dintre cele enumerate mai sus şi un indiciu al activităţii proiectului, de tipul existenţei unor commituri sau versiuni recente. Acest nucleu informativ are rolul de a scădea costul de verificare pentru utilizator şi de a permite o decizie rapidă, fără a presupune parcurgerea întregii documentaţii a proiectului.

Calitatea procesului trebuie să fie evaluabilă prin câteva obiective măsurabile care să nu depindă de detalii de implementare. În mod firesc, se dorește o rată de aprobare după verificarea umană care să fie semnificativă, dar nu maximală, semn că filtrarea automată aduce candidaţi promiţători fără a forţa includerea lor; o ţintă rezonabilă poate fi ca cel puţin două treimi dintre propuneri să fie confirmate într-o perioadă de referinţă. De asemenea, procesul ar trebui să producă un volum stabil de descoperiri, de pildă câteva instrumente noi pe lună, în funcţie de ritmul real al ecosistemului. Completitudinea informaţiilor oferite pentru fiecare intrare poate fi urmărită ca proporţie a câmpurilor obligatorii populate corect, cu aşteptarea de a atinge un prag ridicat şi consistent în timp. În fine, timpul dintre identificare şi verdictul utilizatorului trebuie menţinut scurt, astfel încât recomandările să rămână proaspete şi utile.

În ansamblu, nu se urmărește stabilirea valorii tehnice definitive a fiecărui instrument, ci furnizarea unui mecanism repetabil de descoperire și triere inițială, care aduce în atenție candidați relevanți, atașează informația minimă necesară unei decizii informate și menține controlul la nivel uman acolo unde este necesar.

3 State-of-the-art

State-of-the-art în descoperirea instrumentelor de verificare formală evidențiază un peisaj fragmentat, dificil de cartografiat și lipsit de un director centralizat, în ciuda numărului mare de instrumente existente. Sursele actuale care cataloghează astfel de unelte sunt variate, dar fiecare prezintă limitări importante. Paginile generale, precum categoria „Formal methods tools" de pe Wikipedia sau diversele wiki-uri comunitare, oferă o acoperire redusă și adesea învechită, reflectând rareori aparițiile recente din domeniu. În special wiki-urile dedicate metodelor formale depind aproape integral de contribuții voluntare, iar multe secțiuni păstrează informații din anii 2000, ceea ce le transformă mai degrabă în resurse istorice decât în instrumente actuale de descoperire.

O altă sursă notabilă o reprezintă site-urile organizațiilor profesionale, precum Formal Methods Europe, care întrețin liste structurate de instrumente împărțite pe categorii precum model checking, teoreme-proving sau solvere SMT. Deși aceste liste sunt considerate printre cele mai credibile și bine organizate, ele sunt totuși dependente de contribuțiile dezvoltatorilor, iar instrumentele neanunțate de autori rămân în afara catalogării. Chiar FME recunoaște limitele clasificării pe categorii fixe și sugerează trecerea la un sistem flexibil de etichete, care însă nu este încă implementat. În mod similar, listele comunitare de pe GitHub, de tip „awesome lists", pot fi utile pentru orientare rapidă, dar sunt neoficiale, incomplete și lipsite de rigoare în selecție și actualizare, motiv pentru care nu pot fi considerate surse standardizate.

Pentru anumite subdomenii au apărut portaluri specializate precum Termination-Portal, care oferă liste de instrumente pentru terminarea programelor, incluzând unelte precum AProVE sau TTT2. Cu toate acestea, aceste portaluri suferă de probleme severe de mentenanță; multe secțiuni nu au mai fost actualizate din 2008, ceea ce compromite utilitatea lor în raport cu evoluțiile recente. În analiza complexității programelor situația este și mai fragmentată, multe instrumente fiind cunoscute doar prin participările la competiții sau prin literatura științifică, fără a exista un director care să le centralizeze în mod coerent. În schimb, competițiile internaționale precum SV-COMP oferă surse actualizate și standardizate, întrucât obligă participanții să își documenteze uneltele. Portalul Tools for Formal Methods întreținut de SOSY-Lab conține astfel de informații, incluzând instrumente precum CPAchecker, Ultimate Automizer sau ESBMC, împreună cu detalii despre tehnicile folosite și rezultatele obținute. Totuși, acoperirea acestor resurse se concentrează în principal pe verificarea programelor C și Java, lăsând mai puțin vizibile instrumentele din alte ecosisteme.

În domenii emergente, cum este verificarea rețelelor neuronale, lipsa unor liste centralizate a reprezentat mult timp o dificultate. Apariția VNN-COMP în 2020 a început să suplinească această absență, oferind anual un inventar al instrumentelor precum Marabou, Reluplex sau VeriNet și raportând performanțele acestora. Totuși, nu există încă un portal stabil echivalent QBFLIB, iar evoluția rapidă a domeniului face ca orice listă să devină învechită într-un timp scurt. Spre deosebire de aceste inițiative noi, zona QBF beneficiază de QBFLIB, o bibliotecă matură și bine întreținută de solvere precum DepQBF, CAQE sau

GhostQ. Cu toate acestea, și QBFLIB este dependentă de contribuții voluntare, ceea ce înseamnă că instrumentele necunoscute pot rămâne neincluse.

Descoperirea efectivă a acestor instrumente poate fi realizată prin căutare clasică sau prin interogare asistată de modele de limbaj de tip LLM. Căutarea manuală presupune interogări pe motoare de căutare, consultarea literaturii de specialitate și analiza documentației oficiale a proiectelor. Avantajul acestei metode constă în controlul și acuratețea ridicată a informațiilor, dar procesul este consumator de timp, necesită expertiză și poate omite unelte din afara zonei de vizibilitate a cercetătorului. În schimb, modelele LLM pot oferi rapid liste inițiale și pot agrega informații din surse dispersate, menționând uneori instrumente obscure sau dificil de găsit. Totuși, răspunsurile generate pot conține inexactități, confuzii sau informații învechite, întrucât modelele nu au un mecanism intern de verificare factuală și nici nu garantează exhaustivitatea rezultatelor. În practică, o strategie hibridă, adică generarea unei liste brute cu ajutorul unui LLM, urmată de verificare manuală, este cea mai eficientă.

4 Instalare și configurare

Aplicația este alcătuită din două componente principale: un backend dezvoltat cu FastAPI, responsabil de interogarea unui model LLM (local sau online), normalizarea rezultatelor și gestionarea fluxului de aprobare, și un frontend dezvoltat în React, care afișează instrumentele aprobate și oferă o interfață de revizuire pentru intrările aflate în starea pending. Structura proiectului este organizată astfel încât backendul expune endpoint-uri REST, iar frontendul consumă aceste endpoint-uri printr-un modul dedicat de acces API. Datele persistente (listele pending, approved, denied) sunt stocate local în fișiere JSON, ceea ce simplifică rularea în context educațional și permite testarea fără infrastructură suplimentară.

Pentru rulare sunt necesare un mediu Python (recomandat Python 3.10+), Node.js (recomandat 18+), precum și Ollama pentru execuția locală a modelelor LLM. Instalarea dependințelor backend se realizează prin managerul de pachete pip, pe baza fișierului de cerințe al proiectului, incluzând biblioteci precum fastapi, uvicorn, python-dotenv, jinja2 și ollama. Pentru frontend se instalează dependențele prin npm sau yarn, iar aplicația este pornită în mod uzual pe portul 5173 (Vite). Comunicarea dintre cele două componente este permisă prin configurarea CORS în backend, astfel încât originile locale utilizate în dezvoltare să fie acceptate explicit.

Configurarea backendului se face printr-un fișier .env, din care sunt citite variabilele necesare interogării modelelor și setărilor de execuție. Pentru rularea locală cu Ollama este utilizată variabila OLLAMA MODEL, care specifică tagul modelului instalat local (de exemplu, gemma2:2b). În cazul utilizării unui provider online, proiectul suportă chei de acces și numele modelului prin variabile precum OPENAI API KEY / OPENAI MODEL și GEMINI API KEY / GEMINI MODEL. În practica curentă, fluxul de descoperire se bazează pe Ollama, iar endpoint-ul de sugestii generează candidați în format CSV pe baza

unui prompt parametrizat (data curentă, numărul maxim de rezultate). Prompturile sunt păstrate în directorul prompts, iar backendul le localizează automat prin parcurgerea ierarhiei de directoare pentru a evita dependența de o cale fixă.

Un pas important al configurării este inițializarea spațiului de stocare pentru intrările propuse și validate. Backendul creează automat directorul data și fișierele pending.json, approved.json și denied.json dacă acestea nu există, asigurând astfel persistența minimă a informațiilor între rulări. În plus, pentru a limita erorile produse de rezultate generate de LLM, intrările sunt normalizate înainte de stocare: câmpuri precum numele, descrierea, categoria, licența, limbajele suportate și URL-ul oficial sunt mapate într-un format intern stabil. URL-urile sunt supuse unei validări automate (de exemplu, acceptarea exclusiv a linkurilor HTTPS și respingerea gazdelor nepublice), iar în funcție de configurația implementată pot fi aplicate și verificări de accesibilitate (reachability) pentru a reduce probabilitatea de linkuri invalide sau inventate.

Pornirea aplicației urmează un flux standard de dezvoltare locală. Backendul este lansat prin uvicorn (instanțiind aplicația FastAPI), iar frontendul este pornit separat prin serverul de dezvoltare Vite. Modulul de acces la API din client utilizează variabila VITE API BASE URL pentru a indica adresa backendului (implicit http://127.0.0.1:8000). În acest mod, pagina inițială poate solicita lista de instrumente aprobate, iar pagina de revizuire poate declanșa interogarea LLM-ului și poate opera acțiunile de approve/deny. Separarea clară între pending și approved asigură respectarea cerinței de human check approving: doar intrările verificate și confirmate sunt vizibile pe pagina inițială, restul rămânând disponibile exclusiv pentru revizuire.

5 Statusul implementării

În stadiul curent, aplicația acoperă fluxul principal cerut: obținerea periodică (la cerere, printr-un buton de interogare) a unor instrumente de verificare formală dintr-un model LLM, salvarea lor în zona pending și afișarea exclusiv a instrumentelor aprobate de un utilizator uman. Backendul este implementat în FastAPI și expune endpoint-uri pentru generarea de sugestii prin LLM în format CSV, salvarea și listarea instrumentelor pending, aprobarea/respingerea instrumentelor și listarea instrumentelor approved. Persistența este realizată prin fișiere JSON separate pentru pending, approved și denied, iar inițializarea acestora este gestionată automat la pornirea aplicației.

Componenta de interogare LLM este funcțională prin integrarea cu Ollama, folosind un model local configurabil din fișierul .env. Promptul impune un output strict CSV cu câmpuri standardizate (nume, categorie, descriere, funcționalități, limbaje/modele suportate, licență, actualizare/versiune, URL oficial, scor), ceea ce permite parsarea automată în backend. După parsare, intrările sunt normalizate într-un format intern unitar, iar URL-urile sunt trecute printr-un mecanism de validare (de exemplu, acceptarea exclusiv a adreselor HTTPS și filtrarea gazdelor nepublice) pentru a reduce probabilitatea de linkuri inventate sau invalide.

Interfața frontend, dezvoltată în React, oferă două pagini principale: o pagină inițială ce afișează exclusiv instrumentele aprobate, împreună cu toate metadatele relevante și link către sursa oficială, și o pagină de revizuire care include un buton de interogare a LLM-ului și un tabel unic pentru lista pending, în care fiecare intrare poate fi inspectată (prin details) și apoi aprobată sau respinsă. Fluxul este astfel: interogarea LLM, intrări salvate direct în pending, verificare umană, approve/deny, afișare automată în lista approved. Funcționalitățile cerute (descoperire asistată de LLM + aprobare umană înainte de afișare) sunt astfel deja implementate și utilizabile end-to-end.

6 Arhitectura sistemului

Sistemul este construit pe o arhitectură client–server, împărțită în două componente principale: un frontend web pentru interacțiunea utilizatorului și un backend REST pentru orchestrarea interogării LLM, procesarea rezultatelor și stocarea instrumentelor. Scopul arhitecturii este să implementeze un flux human-in-the-loop în care sugestiile generate automat sunt afișate numai după o verificare și aprobare explicită.

Frontendul este implementat în React și are rolul de a afișa instrumentele aprobate și de a oferi o interfață de revizuire pentru intrările pending. Interfața utilizează un modul unic de acces la API (src/lib/api.js), care încapsulează apelurile către backend și normalizează tratarea erorilor. Clientul apelează endpointul GET /api/tools/approved și redă lista într-un tabel cu metadatele relevante (descriere, funcționalități, licență, limbaje/modele suportate, versiune/actualizare și URL). În pagina de revizuire, utilizatorul poate declanșa interogarea către LLM și poate aproba sau respinge intrările aflate în pending.

Backendul este implementat cu FastAPI și expune endpoint-uri separate pentru funcționalitățile importante ale sistemului. Componenta de interogare (/api/tools/ suggest) construiește promptul și apelează modelul LLM local via Ollama, solicitând un output strict CSV. Rezultatul brut este apoi procesat în două etape: extragerea și parsarea tabelului CSV într-o structură internă și normalizarea câmpurilor într-un format uniform utilizat de aplicație (de ex. name, category, description, url etc.). Această separare permite decuplarea formatului produs de model (CSV) de formatul folosit pentru stocare și afișare.

Persistența datelor este realizată simplu prin fișiere JSON (de exemplu pending.json, approved.json, denied.json), gestionate de modulul toolsStore. La adăugarea în pending, fiecare instrument este normalizat, primește un identificator unic și este înregistrat împreună cu metadate de tip created at și status. Pentru a reduce riscul de intrări eronate, backendul include un pas de validare automată, concentrat în special pe câmpul URL: se acceptă doar linkuri HTTPS și se filtrează gazde nevalide/nepublice, evitând astfel situațiile frecvente în care LLM-ul poate „inventa” adrese sau poate produce linkuri incomplete. În plus, sistemul poate aplica deduplicare (de exemplu după URL) pentru a preveni reintroducerea acelorași instrumente în liste.

Fluxul operațional complet este următorul: utilizatorul apasă butonul de „Query LLM" în pagina de revizuire; frontendul apelează POST /api/tools/suggest; backendul interoghează Ollama și parsează răspunsul; intrările rezultate sunt validate și salvate direct în pending prin endpoint-ul de stocare; apoi pagina de revizuire afișează lista pending, iar utilizatorul aplică decizia de approve sau deny. Aprobarea mută intrarea în approved, iar respingerea o mută în denied; în final, pagina inițială consumă exclusiv lista approved, ceea ce garantează că afișarea rămâne condiționată de verificarea umană.

Această arhitectură este intenționat modulară: componenta LLM poate fi înlocuită cu un serviciu extern (de ex. OpenAI/Gemini) fără a modifica frontendul, iar regulile de validare/normalizare pot fi extinse gradual (ex. verificare de reachability, verificare licență, îmbunătățirea promptului) fără a afecta fluxul de aprobare. În acest mod, sistemul menține un echilibru între automatizare (descoperire rapidă) și control (filtru uman înainte de afișare).

7 Diagrame UML

Diagrama de cazuri de utilizare din fig. 1 surprinde funcționalitățile principale ale aplicației din perspectiva utilizatorului. Acesta poate consulta lista de instrumente aprobate, poate interoga LLM-ul pentru propuneri noi și poate revizui lista de intrări în așteptare. Decizia finală de aprobare sau respingere reprezintă etapa de human check, care controlează ce intră în lista de tool-uri verificate.

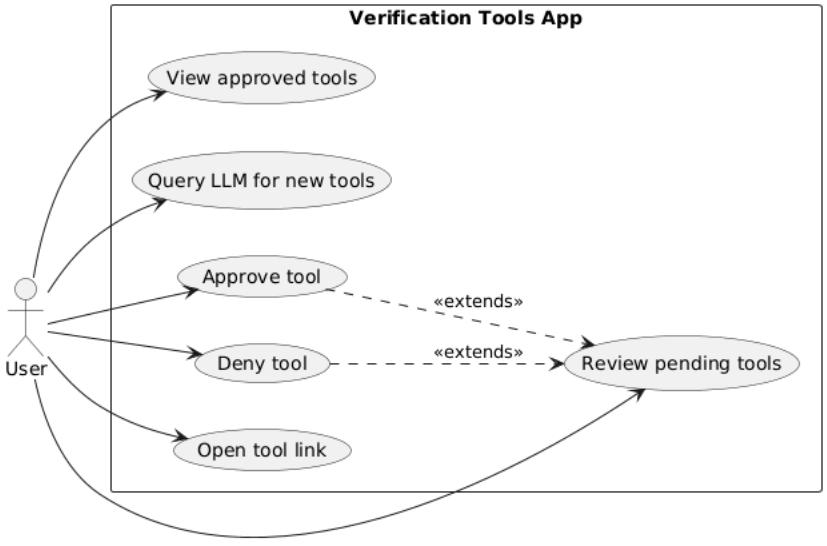

Fig. 1. Cazuri de utilizare ale aplicației

Fluxul detaliat de interacțiune este ilustrat în fig. 2. Secvența arată cum UI-ul declanșează interogarea LLM-ului, backendul parsează rezultatul și normalizează/validează metadatele (în special linkurile), apoi salvează intrările în pending. În continuare, utilizatorul consultă lista de intrări în așteptare și execută acțiuni de approve sau deny, care mută instrumentele în fișierele corespunzătoare și determină ce devine vizibil pe pagina inițială.

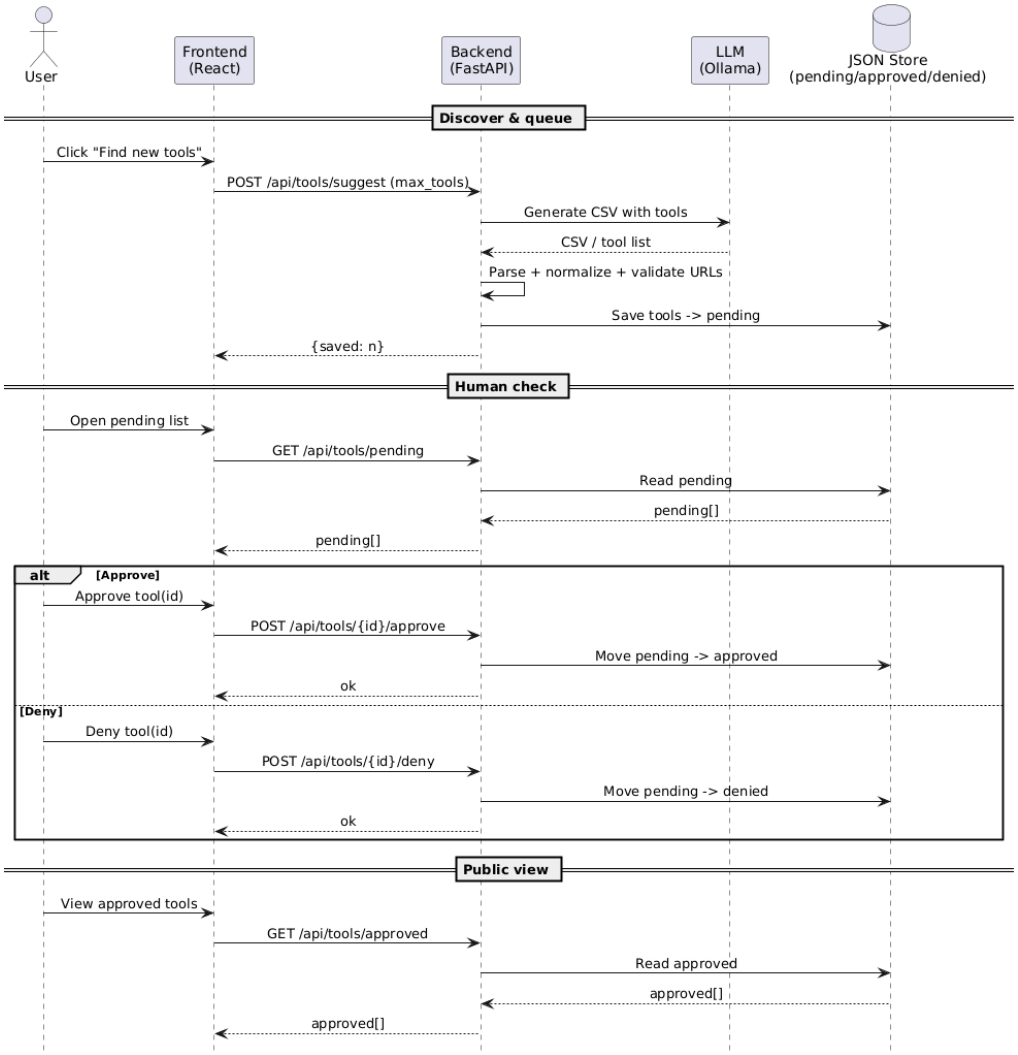

Fig. 2. Fluxul de interacțiune

Diagrama de componente din fig. 3 evidențiază separarea clară între interfața React și backendul FastAPI. Frontendul utilizează un strat de acces (api.js) pentru a apela endpointurile REST, în timp ce backendul include module dedicate pentru sugestii LLM și pentru stocarea/revizuirea instrumentelor. Persistența este realizată prin fișiere JSON (pending/approved/denied), iar interogarea LLM-ului este delegată către runtime-ul Ollama cu un model local.

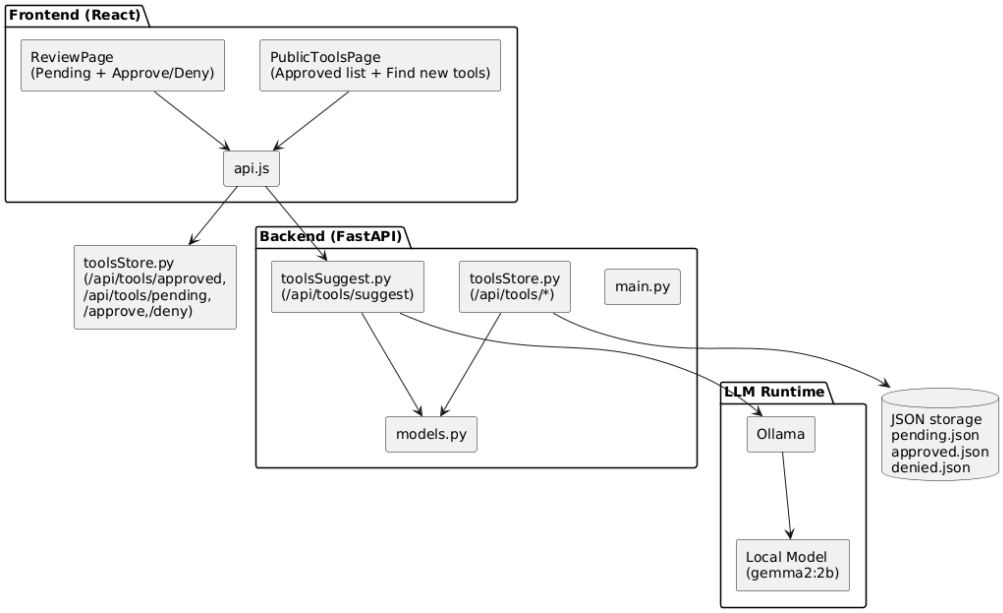

Fig. 3. Componentele aplicației

În final, diagrama de deployment din fig. 4 descrie modul de execuție al sistemului în mediul local. Aplicația React rulează în browser și comunică prin HTTP cu serverul FastAPI (Uvicorn), care gestionează logica de business și citirea/scrierea fișierelor JSON. Pentru generarea propunerilor, backendul comunică cu serviciul Ollama (local), astfel încât întregul flux (descoperire, validare automată de bază și revizie umană) poate fi rulat fără dependențe externe.

8 Link Github

Toate fișierele asociate proiectului, inclusiv codul sursă și componentele frontend și backend ale aplicației, pot fi accesate public la adresa:

https://github.com/tsonicm/ProiectVF

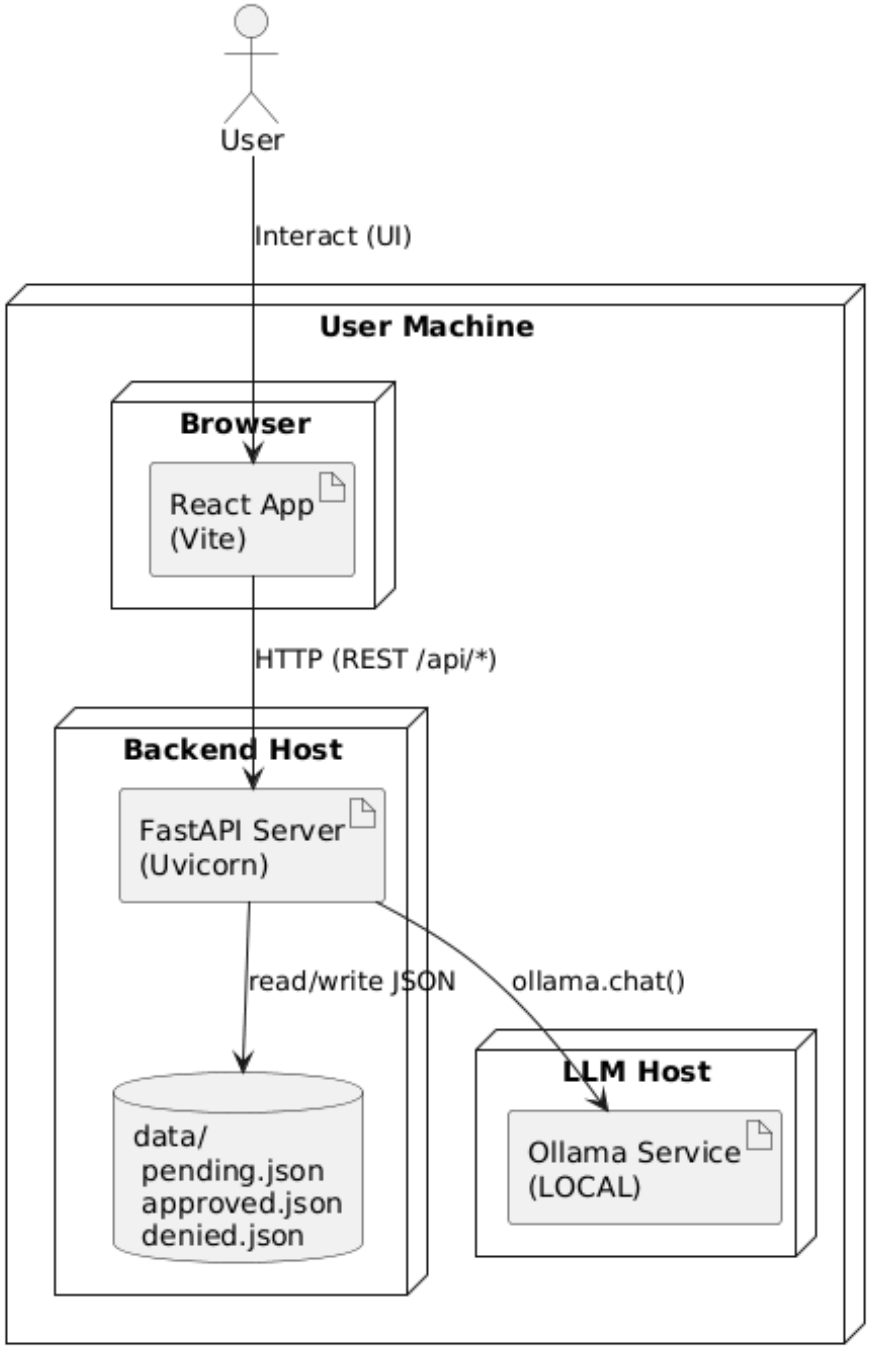

Fig. 4. Arhitectura de execuție

9 Dificultăți întâmpinate

În procesul de dezvoltare, una dintre dificultățile principale a fost legată de calitatea metadatelor generate de modelul LLM. Deși multe dintre numele de instrumente propuse erau reale și descrierile păreau plauzibile, modelul avea tendința să genereze uneori URL-uri inventate sau greșite (de exemplu, domenii inexistente, pagini care nu aparțin proiectului sau linkuri care nu mai sunt valide). Această problemă a afectat direct utilitatea listei pending, deoarece un link incorect îngreunează verificarea umană și poate introduce intrări eronate în catalog. Pentru a reduce acest risc, a fost necesară rafinarea promptului prin reguli explicite („nu inventa URL-uri", „folosește doar surse oficiale HTTPS") și introducerea unor verificări suplimentare în backend, astfel încât intrările cu linkuri nevalide să fie filtrate sau respinse înainte de stocare.

O a doua dificultate importantă a apărut la rularea locală a modelelor prin Ollama, unde resursele hardware au devenit un factor limitativ. Modelele mai mari, care ar putea produce rezultate mai consistente și mai puține halucinații, necesită cantități considerabile de memorie RAM, iar în mediul de testare acest lucru a dus la erori de execuție, instabilitate sau imposibilitatea de a încărca anumite modele. Ca urmare, a fost nevoie de un compromis între acuratețe și fezabilitate: folosirea unui model mai mic, care rulează stabil local, dar care poate necesita filtrare mai strictă și o validare mai atentă a rezultatelor. În practică, această constrângere a influențat atât selecția modelului utilizat, cât și designul sistemului, accentul mutându-se pe validare automată de bază și pe rolul verificării umane înainte de afișare.

10 Rezultate

Rezultatul principal al proiectului este obținerea unui flux repetabil de descoperire și triere inițială a instrumentelor de verificare formală, adaptat unui ecosistem în care apar frecvent proiecte noi, iar informațiile relevante sunt dispersate și greu de menținut actualizate. Prin interogarea unui model LLM pe categorii țintă (corectitudine funcțională, terminare, limite de complexitate, verificarea rețelelor neuronale și solvere QBF), sistemul generează o listă de candidați însoțiți de metadate minimale (nume, descriere, funcționalități, limbaje/modele suportate, licență, versiune/actualizare și URL oficial). În acest mod, procesul reduce semnificativ costul de căutare manuală și oferă un punct de plecare coerent pentru evaluare.

Un alt rezultat important, specific domeniului verificării formale, este creșterea utilității practice a listei generate prin introducerea unei etape de control uman. În practică, un catalog public devine cu adevărat valoros doar dacă intrările sunt verificabile și trasabile către o sursă oficială. Tocmai de aceea, sistemul separă clar propunerile: intrările returnate de LLM sunt salvate în pending și devin vizibile doar după o decizie explicită de aprobare. Această etapă funcționează ca filtru de încredere, reducând efectele tipice ale halucinațiilor (de exemplu, linkuri inventate sau proiecte confundate) și menținând rigoarea necesară într-un context de cercetare și experimentare.

Din perspectiva impactului asupra activității de explorare a tool-urilor, aplicația accelerează faza de "scouting" și scurtează timpul până la o listă scurtă de candidați care merită testați. În locul căutărilor repetate pe web și al consultării manuale a mai multor surse, utilizatorul poate declanșa interogarea și poate primi rapid propuneri structurate, comparabile între ele. Metadatele standardizate facilitează decizii rapide legate de potrivirea unui instrument cu un scenariu (de exemplu, limbaje suportate, tipul analizei, disponibilitatea unui repository și indiciul de mentenanță prin versiune/actualizare), ceea ce este util atât pentru cercetători, cât și pentru practicieni care vor să evalueze alternative.

În final, rezultatul proiectului este un mecanism end-to-end funcțional care susține în mod pragmatic actualizarea periodică a unei liste de instrumente verificate. Prin faptul că outputul LLM este cerut într-un format strict (CSV), iar backendul normalizează câmpurile într-o structură stabilă, sistemul poate integra ușor îmbunătățiri ulterioare fără a schimba fluxul principal: de exemplu, rafinarea promptului pentru a reduce linkurile inventate, întărirea validării automate a URL-urilor sau înlocuirea modelului local cu un provider extern. Astfel, proiectul oferă o bază solidă pentru menținerea unui catalog mai proaspăt și mai coerent, păstrând totodată controlul calității prin verificare umană.