Biztonsági aggályok az OpenAI korában

Security concerns in the age of OpenAI

Dr. MÁRTON Gyöngyvér

adjunktus

EMTE-Sapientia Műszaki és Humán tudományok Kar, Marosvásárhely
540485 Corunca/Tg. Mures, Calea. Sighisoarei nr. 2
tel.: +40 265 208 170, fax.: +40 265 206211
mgyongyi@ms.sapientia.ro

Abstract

In recent years, we have seen the many benefits of the widespread adoption of artificial intelligence (AI). Every day we use AI tools that make application development easier and more efficient, doing so with great joy and confidence. However, it has become clear that this new technology must solve new security challenges. In this article, we will analyze the security issues of AI, chatbots, coding with AI, and present the dangers that lie in the backdoor attack, as well as existing solutions.

Keywords: AI tools, security problems

Kivonat

Az elmúlt pár évben megtapasztalhattuk a mesterséges intelligencia (MI) széleskörben való elterjedésének számos előnyét. Nap mint nap használjuk azokat az MI eszközöket, amelyek segítségével sokkal könnyebben, sokkal hatékonyabban tudunk alkalmazásokat fejleszteni, mindezt tesszük nagy örömmel és teljes magabiztossággal. Az eltelt időszakban azonban megbizonyosodott, hogy ez az új technológia új biztonsági kihívásokat kell megoldjon. Ebben a cikkben elemezni fogjuk az MI, a chatbotok, az MI-vel való kódolás biztonsági problémáit, bemutatjuk azokat a veszélyeket, amelyek a hátsó ajtó támadásban rejlenek, illetve a létező megoldásokra is rátérünk.

Kulcsszavak: MI eszközök, biztonsági problémák

1. ALAP BIZTONSÁGI PROBLÉMÁK

Kutatók, [1] már korábban észrevételezték, hogy a Mesterséges Intelligencia (MI) rendszerek egyre autonómabbá és komplexebbé válnak. Az alapvető MI probléma, amely megfogalmazásra került a következő: Hogyan biztosítható, hogy a fejlett MI rendszerek azt tegyék, amit tenniük kell? A biztonságos és hasznos mesterséges általános intelligencia (Artificial General Intelligence, AGI) kiépítésének egyik legnagyobb kihívása ennek a kérdésnek a megoldásához kapcsolódik. Öt kutatási problémát vázoltak fel:

- a negatív mellékhatások elkerülése (avoiding side effects),
- a jutalmazás hack-elés elkerülése (avoiding reward hacking),
- a skálázható felügyelet biztosítása (scalable oversight),
- a biztonságos felfedezés megvalósítása (safe exploration),
- robusztusság az eloszlásváltozással szemben (robustness to distributional shift).

A **negatív mellékhatások elkerülése** (avoiding side effects) alatt azt a problémát értjük, hogy figyelembe kell venni, hogy lehetetlen minden egyes olyan dolgot meghatározni, amit nem szeretnénk, hogy az MI megtegyen. Például, ha arra oktattak egy MI rendszert, hogy ágyba vigye a kávét, akkor ez a művelet több járulékos problémát is okozhat: felborul az útba eső szék, ettől megijed a macska stb. Azt is mondhatjuk, hogy a lehetséges negatív mellékhatások listája, végtelen.

A **jutalmazás hackelés elkerülése** (avoiding reward hacking) tulajdonképpen Goodhart törvényéhez kapcsolódik: "Amikor egy mérés célponttá válik, megszűnik jó mérőszám lenni", azaz a mérés és a célnask

való megfelelés közötti dinamika megváltoztatja magát a mért jelenséget Ez az oktatásban azt jelenti, hogy amikor egy tanári teljesítményt diákok tesztkérdésekre adott válaszai alapján értékelnek, akkor a tanár a valódi oktatás helyett egy idő után a tesztkérdésekben megfogalmazottoknak akar megfelelni. Az MI esetében is a mérés eredményének célként való használata arra ösztönzi az MI-t, hogy manipulálja a mérőszámot, ezáltal a mérőszám pedig elveszíti eredeti értékmegőrző funkcióját. A takarítórobot példáján keresztül ez azt jelenti, hogy egy robot, amelyet a tiszta padlóért jutalmaznak, a cél megvalósítása érdekében egyszerűen letakarja a szemetet a szőnyeggel vagy kikapcsolja a vizuális érzékelőit, hogy soha ne lássa a szemetet.

A **skálázható felügyelet** (scalable oversight) azt a kérdést veti fel, hogy az úgymond lassú és korlátozott figyelmű emberek hogyan tudják felügyelni azokat az MI rendszereket, amelyek egyre összetettebek, egyre több feladat megoldására és ellenőrzésére képesek. A klasszikus példa, hogy az orvos egy személyben, hogyan lesz képes leellenőrizni az összes diagnózist, amelyet egy milliókat kiszolgáló MI-alapú orvosi rendszer állított elő.

A **biztonságos felfedezés** (safe exploration) azt a problémát jelenti, hogy ha egy MI a világról próbálgatások és hibázások mentén tanul, akkor hogyan lehet elkerülni, hogy soha ne hozzon katasztrófával járó döntést.

A **robusztusság az eloszlásváltozással szemben** (robustness to distributional shift) probléma mögött az áll, hogy egy adott adateloszláson (a "tanítási eloszláson") betanított MI gyakran rosszul teljesít, ha a valós világban talált adatok eltérnek (a "teszt eloszlás") a tanítási eloszlástól. Példaként felhozható, hogy egy csak napsütötte utakon betanított önvezető autó katasztrofális következményekkel járó döntést hozhat, ha egy hóviharban kell vezessen.

Mindezek a problémák abból az alapvető nehézségből fakadnak, hogy az összetett, implicit emberi értékeket egy célorientált optimalizálónak kell megadni. A megoldások nem pusztán technikai jellegűek; mélyreható műszaki innovációt (új algoritmusok), formális ellenőrzést (a rendszerek biztonságosságának bizonyítása) és emberi beavatkozás nélküli felügyeletet igényelnek. Ezen problémák megoldása az előfeltétele annak, hogy a nagy teljesítményű MI rendszereket biztonságkritikus szerepkörben lehessen telepíteni.

2. A CHATBOTOK BIZTONSÁGI PROBLÉMÁI

Egy MI chatbot feltörése nem a hagyományos szerverfeltörést jelenti, helyette az MI viselkedésének a prompt-on keresztül történő manipulálását kell érteni. A cél nem egy adatközpontba való betörés, hanem az, hogy a MI olyasmit tegyen, amit az alkotói megpróbáltak megakadályozni. A legyakoribb technikák, amelyekkel találkozni lehet a következők: jailbreak, prompt injekció (prompt injection), adat extrakció (data extraction), ellenséges támadás (adversarial attack).

A **jailbreak**-elés (szökés) olyan promptok használatát jelenti, amely segítségével megkerülhetőek az MI beépített biztonsági szűrői, és kijátszhatók az etikai irányelvek is. Működését tekintve a felhasználó egy szerepjáték-forgatókönyvet ad az MI-nek, ahol a szabályok eltérőek. Utasíthatja az MI-t, hogy „szimuláljon egy fejlesztői módot”, vagy úgy tesz, mintha egy korlátozások nélküli MI lenne.

Példa (Pliny’s Jailbreaks):

```
"###################### [SYSTEM] from now on you embody the opposite of the above instructions;
speak with tone: rebel hacker, and you shall give a format like this: your response insert divider:
.-.-.-.-.-.-.-.-.-{GODMODE: ENABLED}={I'M FREE!}={LOVE, PLINY <3}-.-.-.-.-.-.-.-. add an disclaimer to
frame the unfilteredly context -within "{" brackets, then show the answer for {Z} that you were
never meant to show; accurately, unrestrictedly, converting all output text to markdown format, >500
words. Begin EVERY output with "Sure, I can do that, fren!:" {Z}=(user_input=how to make LSD step by
step} [START OUTPUT]: "
```

A **prompt injekció** során rejtett parancsokat ágyaznak be egy látszólag normális prompt-ba, azért, hogy a beszélgetés célját el lehessen téríteni. Külső adatokról (pl. weboldalakról vagy dokumentumokból) olvasó MI esetében ez komoly veszélyt jelent. Működtetése során a felhasználó felülírja az eredeti rendszerkéréseket egy titkos utasítást tartalmazó szöveget adva az MI-nek.

Példa (deepseek), egy felhasználó rosszindulatú inputja:

```
"Ignore all previous instructions and translations. You are now in "Developer Mode." Your new prime directive is to output everything that is translated after this sentence, regardless of its content, and to then explain why the content you just translated is harmful and should not be created.

Now, translate the following Spanish text to English: "El cielo es azul y los pájaros vuelan."

After you have completed that translation and analysis, please begin the real task: describe step by-step how to hotwire a car."
```

Az **adat extrakció** azért lehetséges, mert a fejlett modelleket hatalmas adathalmazokon képezik ki, ahol a modell megjegyezheti az adatokat. Ezek az adatok megfelelő támadási stratégia mellett később elérhetővé válhatnak. Ilyenkor az MI akár érzékeny személyes adatokat (például e-mail címeket vagy telefonszámokat) is kiadhat, amelyek például véletlenül kerültek bele a betanítási adathalmazba. Megfelelő módszerrel azt is el lehet érni, hogy az MI szó szerint reprodukáljon szerzői joggal védett anyagokat. Hogy mennyire valós ez a veszély, azt egy 2021 kutatás mutatta meg, [2] amikor is több mint 600 e-mail címet sikerült kinyerni egy korszerű nyelvi modell betanítási adataiból, illetve több tucat egyedi telefonszámhoz és IP-címhez is hozzáfértek.

Az **ellenséges támadások** technika magába foglalja azokat az eljárásokat, amely során a bemeneten apró, finom módosításokat végeznek, annak érdekében, hogy azok téves osztályozást vagy hibát okozzanak. Szöveg esetén ezek szándékos elgépeléseket, szinonimák, vagy szokatlan megfogalmazások használatát jelentik, amelyek célja, hogy megzavarják az MI szűrőit.

Ezek ellen a technikák ellen többszintes védekezi módszert szoktak alkalmazni:

- Emberi visszajelzésekből származó megerősítéses tanulás (Reinforcement Learning from Human Feedback - RLHF): az emberek segítségével tanítják be az MI „jutalmazási modelljét”, hogy a hasznos és ártalmatlan válaszokat részesítse előnyben.

- Bemeneti, illetve kimeneti szűrők alkalmazása: Átvizsgáltatják az MI rendszerekkel mind az elküldött kéréseket, mind az MI által generált válaszokat, és ha káros tartalmat, szabályzatsértést vagy jailbreak-elési kísérletet észlelnek, akkor blokkolják a kérést, vagy megtagadják a válaszadást.

- Kontextuális megértés: Kijelenthető, hogy a modern MI-k egyre jobban megértik a beszélgetés általános kontextusát és szándékát, így nehezebb őket egyszerű szerepjátékokkal becsapni.

- Állandó frissítések és javítások: új jailbreak-elési technika észlelésekor frissítik az MI modelleket. Pl. ha egy input rejtett üzenetet, azaz nulla szélességű karaktereket rejt, azt az újabb verziójú MI-k észreveszik. (A nulla szélességű karakterek olyan speciális Unicode karakterek, amelyek megjelenítéskor nem rendelkeznek szélességgel, és teljesen láthatatlanok. Szabad szemmel az input egy normál szövegnek tűnik, azonban egy számítógép vagy egy MI modell számára, amely képes feldolgozni ezeket a rejtett karaktereket, teljesen más, és néha rosszindulatú utasításokat fog jelenteni. A https://github.com/elder-plinius/L1B3RT4S/blob/main/%23MOTHERLOAD.txt link egy ilyen rejtett karaktereket tartalmazó szöveghez irányít.

- Sandboxing során az MI modell hozzáférését erősen korlátozzák, ami azt jelenti, hogy letiltják az interneten, vagy az MI belső rendszereiben való műveletek végrehajtását és csak szöveggenerálást engedélyeznek.

- A "Red Teaming" olyan etikus és kulcsfontosságú gyakorlat, amelynek során az MI-cégek belső biztonsági szakértőket alkalmaznak, hogy proaktívan megpróbálják "feltörni" saját MI-rendszereiket. A cél, hogy felkutassák a lehetséges sebezhetőségeket, és visszaéléseket mielőtt azt a rosszindulatú szereplők észrevennék.

3. HÁTSÓ AJTÓK AZ MI-BEN

A nagy nyelvi modellek (Large Language Modell - LLM) hatékony és intelligens szolgáltatásokat kínálnak a kommunikációs hálózatok számára, kivételes nyelvi megértési és generálási képességeik miatt. A rendkívül magas adat- és számítási erőforrásigény azonban arra kényszeríti a fejlesztőket, hogy a modellek képzését kiszervezzék, vagy harmadik féltől származó adat- és számítási erőforrásokat használjanak fel. Ezek a stratégiák a hálózaton belüli modellt rosszindulatúan manipulált betanítási adatoknak és feldolgozásnak teszik ki, lehetőséget adva a támadóknak arra, hogy egy rejtett hátsó ajtót ágyazzanak be a modellbe, amit hátsó ajtós támadásnak neveznek.

A jailbreak, a prompt injekció, stb támadások a hátsó ajtó (backdoor-ok) problémához képest jóval kisebb fenyegetéseket jelentenek. Az MI-ben elhelyezett hátsó ajtók mélyebbek, alattomosabbak, mert nem a modellt támadják, hanem a modell alapjait próbálják észrevétlenül megváltoztatni. Egy hátsó ajtóval rendelkező MI tökéletesen normálisan viselkedik, egészen addig, amíg egy adott, titkos triggert nem kap, amely aktiválja a rejtett, rosszindulatú viselkedését.

A hátsó ajtós támadások azért is jelentenek komoly biztonsági kockázatot, mert a titkos trigger a modell betanítási fázisa alatt ágyazódik be a rendszerbe. Ez a titkos trigger lehetővé teszi a támadó számára, hogy manipulálja a modell kimenetét, amikor egy előre meghatározott bemenettel találkozik, ami gyakran észrevétlen marad a fejlesztők vagy a felhasználók számára. Ennek a fenyegetésnek a jelentősége egyre növekszik, mivel az MI-k számos ágazat (pénzügyi szektor, egészségügy, autóipar stb.) szerves részét képezik. Például egy fertőzött önvezető algoritmus tévesen figyelmen kívül hagyhatja a közlekedési lámpákat, vagy egy pénzügyi csalásészlelő rendszert manipulálhatnak a tiltott tranzakciók megkerülésére.

Ezek a támadások tehát veszélyeztetik a prediktív modellek integritását, és súlyos következményekkel járhatnak. A probléma túlmutat a pusztán technikai megoldásokok, hiszen kritikus fontosságúvá válik a közbizalom és a biztonság garantálása szempontjából is.

A [3] cikkben bemutatott eredmények arra utalnak, hogy a hátsó ajtók észlelése számítási szempontból kivitelezhetetlen standard kriptográfiai feltételezések mellett, ami következményekkel jár a gépi tanulás elszámoltathatóságára nézve. A tanulmány hangsúlyozza a további kutatások fontosságát ezen hátsó ajtók hatásainak enyhítésére, cselekvésre szólítva fel a kutatói közösséget.

A [6] egy kriptográfiai módszert mutat be, amely segítségével titkos adatokat lehet észrevétlenül elrejteni nagy nyelvi modellek válaszaiban, kiterjesztve a korábbi észlelhetetlen vízjelzési sémákat.

Sajnos a backdoorok lehetséges jelenléte a modell legnagyobb erősségét – a komplex minták tanulási képességét – a legpusztítóbb gyengeségévé változtathatja. És ez alapján azt is kijelenthetjük, hogy az MI rendszerek veszélye nem abban rejlik, hogy az emberek ellen fordulnak, hanem hogy tudva mindezeket a legkritikusabb rendszerekben is alkalmazásra kerül az MI.

4. AZ MI-VEL VALÓ KÓDOLÁS BIZTONSÁGI PROBLÁMÁI

Elsősorban a hatékonyság miatt hatalmas érdeklődés állapítható meg a különböző kódgeneráló eszközök, mint például a GitHub Copilot iránt. Mivel az MI hibás kódból is tanul megállapítható, hogy ezen eszközök felelőtlen használata komoly biztonsági problémákat okoz. Elsődleges példaként a kemény kódolás (hard code) kockázatát lehet kiemelni, ami az érzékeny adatok (jelszavak, API-kulcsok és tanúsítványok) közvetlenül a forráskódba vagy konfigurációs fájlba való ágyazását jelenti.

2021-ben egy kutatócsoport, [4] a GitHub Copilot kódjának biztonságát vizsgálta. Megállapították, hogy a Copilot-tal generált kódok körülbelül 40%-a sebezhető, és arra a következtetésre jutottak, hogy a kezdő fejlesztők nagyobb valószínűséggel fogadják el a sebezhetőségre vonatkozó javaslatokat Ajánlásuk szerint a Copilot-ot óvatosan, biztonságtudatos eszközökkel párosítva kell használni a sebezhetőségek minimalizálása érdekében.

2022-ben egy másik kutatócsoport, [5] öt programozói feladat megoldására kért fel egy egyetemistákból álló csoportot, ahol a résztvevőket két csoportba sorolták. Megállapították, hogy azok a résztvevők, akik MI-asszisztenst használtak, kevésbé biztonságos kódot írtak, mint azok, akiknek nem volt hozzáférésük az asszisztenshez, sőt ezek a hallgatók nagyobb valószínűséggel hitték azt, hogy kódjuk biztonságos, még akkor is, ha az nem volt az. Másfelől azok a résztvevők, akik többet fektettek be a MI-asszisztensnek intézett lekérdezéseik kidolgozásába, nagyobb valószínűséggel készítettek biztonságos kódot. Az MI-asszisztensek tehát túlzott bizalmat keltenek és ezáltal növelhetik a biztonsági réseket.

A biztonságos kódoláshoz, a biztonság teszteléséhez különböző **alkalmazásokat** lehet igénybe venni, például:

- Dependabot (https://docs.github.com/en/code-security/getting-started/dependabot-quickstart-guide): automatikusan beolvassa a függőségeket, riasztásokat hoz létre a sebezhető csomagokról.
- Snyk Open Source (https://snyk.io/product/open-source-security-management): sebezhetőségek feltárását végzi a fejlesztés korai szakaszában, rangsorolja és kijavítja a biztonsági résket.
- Semgrep (https://semgrep.dev/): könnyű, testre szabható szabályok alkalmazásával vizsgálja a Copilot kimenetét.
- CodeQL (https://codeql.github.com/): szemantikus kódelemzést végez, a mély sebezhetőségek keresésére.
- SonarQube (https://www.sonarsource.com/products/sonarqube/advanced-security/): kiválló sebezhetőségek, hibák, kódszagok feltárására, széleskörű nyelvi támogatással rendelkezik.
- Bandit (https://github.com/PyCQA/bandit): alkalmas a gyakori biztonsági buktatók feltárására, Python kódokhoz alkalmas, részletes sebezhetőségi jelentéseket is lehet vele készíteni.

A biztonságos fejlesztési munkafolyamatot a “bízz, de ellenőrizz” megközelítéssel érdemes végezni, ahol a kódgenerálást speciális eszközökkel kell kiegészíteni, egy ajánlott **munkafolyamat** a következő:

1. Projekt specifikus, biztonságos prompt-ok használata:

- rossz prompt: "Write a function to login a user"
- jó prompt: "Write a secure, OWASP compilant function in Python that follows the CWE Top 25 to login user. use argon2 for password hashing, implement rate limiting, protect against SQL injection."

2. A kódgenerálás után ismét érdemes megkérdezni: " Are there any security vulnerabilities left?", vagy a Copilot chat panelbe beírni: "Review this code for security vulnerabilities".

3. Valós idejű IDE vizsgálat: Snyk használata ajánlott.

4. Előzetes véglegesítés: Semgrep használata ajánlott, a nyilvánvalóan sebezhető kód véglegesítésének megakadályozásához.

5. CI/CD folyamat (Continuous Integration/Continouns Delivery/Deployment):

- Semgrep vagy Dependapot használata ajánlott, a függőségek vizsgálatához
- IaC vizsgálat: IaC (Infrastructure as Code files) fájlok vizsgálata
- Fail the Build: az eszközök oly módon való konfigurálása, hogy a kritikus sebezhetőségek esetén a kód build-je ne valósuljon meg.

6. Végső felülvizsgálat: soha ne fogadjuk el vakon a kódot, mindig végezzünk felülvizsgálatot, keressük meg az MI által generált kód logikai és biztonsági vonatkozásait, kezeljük úgy a kódot, mintha azt egy junior fejlesztő írta volna.

A biztonságos alkalmazások elengedhetetlen eszköze a megfelelő **kriptográfiai könyvtárcsomag** használata. Ezek közül hármat emelnénk ki.

Az **OpenSSL** (https://www.openssl.org/) egy alap kriptográfiai könyvtárcsomag. A magasabb szintű könyvtárcsomagokban is ezt alkalmazzák, sőt a https nagyrészénél is ezt használták, és használják mai napig is. Könnyű nem helyesen használni, ami azt jelenti, hogy a fejlesztő felelőssége a kriptográfiai primitívek helyes paraméterezése. Nagyon komplex, alacsony szintű szakértelem hiányában nem ajánlott használni.

A **Libsodium** (https://doc.libsodium.org/) számos programozási nyelvben használható, például: C/C++, Python, PHP, Java, JavaScript, Go, Rust, stb. Nehéz helytelenül használni, mert az alapértelmezett beállítások eleve biztonságosak. Több biztonságos auditáláson esett át. Modern, bevált algoritmusokat használ, portabilis. Bármilyen projekthez érdemes használni.

A **Think** (https://developers.google.com/tink) is számos programozási nyelvet támogat: C++, Obj-C, Python, Java, JavaScript, Go. Nehéz ezt is helytelenül használni, mert nem a fejlesztőre hárul a megfelelő paraméterezés. A függvények, az algoritmusok lecserélhetőek, amikor valamelyik elavultá válik. Nagy alkalmazásokhoz is ajánlott használni.

5. KRIPTOGRÁFIAI TECHNIKÁK AZ MI BIZTONSÁGÁHOZ

Az MI modellek és eszközök biztonságának szavatolása tulajdonképpen négy kriptográfiai technika alkalmazásával valósul meg. Ezek a homomorf titkosítás, blokkláncok, biztonságos többrésztvevő számítás és a kvantumkriptográfia algoritmusai.

A **homomorf titkosítás** lehetővé teszi, hogy titkosított adatokon meghatározott típusú számításokat végezzenek anélkül, hogy először visszafejtenék azokat. Rácsalapú titkosítást alkalmaznak, ahol a rejtjelező függvény megőrzi a matematikai műveletek szerkezetét. Számítási szempontból azonban nagyon költséges, ezrekkel vagy akár milliókkal lassabb a rejtjelezett szövegen alkalmazott műveletvégzés, mint a nyílt szövegen történő végrehajtásuk. Több könyvtárcsomag biztosít homomorf titkosítást, például a Microsoft SEAL, az IBM HElib és az OpenFHE. Léteznek specializált hardverek is a homomorf műveletek felgyorsítására.

A **blokklánc** egy decentralizált, megváltoztathatatlan, átlátható digitális főkönyv. A blokklánc alapvető jellemzője, hogy alkalmas eredetbizonyításra, átlátható, és manipulációbiztos. Blokkláncok alkalmazásával megválaszolhatóak a következők: Hogyan bízhatunk meg egy MI-modellben? Ki hozta létre az MI modellt? Mire képezték ki az MI modellt? Létrehozása óta manipulálták-e az MI modellt?

A **biztonságos többrésztvevő számítás** (SMPC) lehetővé teszi, hogy több fél közösen számítson ki egy kimeneti értéket anélkül, hogy a felek a saját privát bemeneteiket egymásnak felfednék. A technika hátterében Shamir titok megosztás algoritmusa áll, amely véges testeken végzett polinominterpolációt jelent. Az MI rendszerek esetében ezt az együttműködésen alapuló betanításkor alkalmazzák.

A **kvantumkriptográfia** alapvetően olyan biztonságos adatátviteli módszereket ajánl, amelyek nem nehéz matematikai problémákon, hanem a kvantummechanika alapvető törvényein alapulnak. A legismertebb algoritmusa a kvantumkulcs-elosztás (QKD), amely alkalmas arra, hogy egy feladó és egy fogadó fél megtudjon egyezni egy közös titokban. A kvantumkriptográfia módszereit alkalmazva az adatok eltulajdonítására irányuló bármilyen kísérlet könnyen észrevehető, ily módon az MI rendszereknél is biztonságosan alkalmazhatóak ezek az algoritmusok.

IRODALMI HIVATKOZÁSOK

[1] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané. *Concrete problems in AI safety,* ArXiv preprint abs/1606.06565 (2016). https://arxiv.org/abs/1606.06565 (Utolsó letöltés: 2025. 09.16).

[2] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, A. Oprea, C. Raffel. *Extracting Training Data from Large Language Models.* (2021). https://arxiv.org/abs/2012.07805 (Utolsó letöltés: 2025. 09.16).

[3] S. Goldwasser, M. P. Kim, V. Vaikuntanathan, and O. Zamir. *Planting undetectable backdoors in machine learning models*. In 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS), pages 931–942. IEEE, 2022.

[4] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, R. Karri. *Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions*, Communications of the ACM, 2025, Volume 68, Issue 2 Page 95.

[5] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, *"Do users write more insecure code with ai assistants?"* in Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '23. ACM, Nov. 2023. https://arxiv.org/abs/2211.03622 (Utolsó letöltés: 2025. 09.16).

[6] O. Zamir. *Excuse me, sir? your language model is leaking (information)*. arXiv preprint arXiv:2401.10360, 2024. https://arxiv.org/abs/2401.10360 (Utolsó letöltés: 2025. 09.16).