UDC: 004.8 Info M str. 4-10

ОПТИМИЗАЦИЈА УПИТА ВЕЛИКИХ ЈЕЗИЧКИХ МОДЕЛА КОРИШЋЕЊЕМ DSPY ОКВИРА
OPTIMIZATION OF LARGE LANGUAGE MODEL PROMPTS USING THE DSPY FRAMEWORK

Тара Милојковић

РЕЗИМЕ: Велики језички модели показују висок степен осетљивости на формулацију упита, због чега ручни *prompt* инжењеринг често даје нестабилне и трошковно неефикасне резултате. DSPy оквир нуди програмски приступ у ком се упит (енг. *prompt*) посматра као параметар који се може систематски учити и оптимизовати. Циљ рада је да се испита у којој мери аутоматска оптимизација упита у DSPy-у побољшава класификацију злонамерних упита у односу на ручно дефинисане упите, уз ограничен буџет и строго дефинисан формат излаза. За потребе експеримента коришћени су јавни скуп података „jailbreak-classification“, модел GPT-4.1-nano и DSPy програм заснован на комбинацији `Signature` декларација и модула `dspy.Predict`, са MIPROv2 оптимизатором и варијантама SIMBA и COPRO. Прати се више показатеља учинка: макро-, тачност, ризик (асиметрична цена FN/FP грешака) и стопа нарушавања формата, као и утицај појединачних компоненти кроз аблациону анализу. У односу на базну линију (макро- = 0,248), DSPy без оптимизације подиже учинак на , док MIPROv2 достиже (празан контекст) и (истинит контекст), врло близу горње границе од . Ризик се значајно смањује, а нарушавање формата је приближно . Аблација показује да прецизније преформулисање упита даје нешто већи појединачни допринос од самих примера, док њихова комбинација остварује најбољи резултат уз умерено већи ризик. DSPy на тај начин претвара дизајн упита у репродуктиван, мерљив и трошковно свестан процес, што је посебно важно у безбедносно осетљивим продукционим системима заснованим на великим језичким моделима.

КЉУЧНЕ РЕЧИ: велики језички модели, prompt инжењеринг, DSPy, оптимизација упита, класификација злонамерних упита.

ABSTRACT: Large language models are highly sensitive to prompt formulation, so manual prompt engineering often yields unstable and cost-inefficient results. The DSPy framework offers a programmatic approach in which the prompt is treated as a parameter that can be systematically learned and optimized. The goal of this paper is to examine to what extent automatic prompt optimization in DSPy improves malicious-prompt classification compared to manually designed prompts, under a constrained budget and a strictly defined output format. For the experiments, the public “jailbreak-classification” dataset, the GPT-4.1-nano model and a DSPy program based on a combination of `Signature` declarations and the `dspy.Predict` module are employed, together with the MIPROv2 optimizer and its SIMBA and COPRO variants. Several performance indicators are monitored: macro-F_1, accuracy, risk (asymmetric FN/FP error cost) and the rate of output-format violations, as well as the contribution of individual components through ablation analysis. Compared to the baseline (macro-F_1 = 0.248), plain DSPy without optimization increases performance to 0.689, whereas MIPROv2 reaches 0.878 (empty context) and 0.936 (truthful context), very close to the upper bound of 0.955. Risk is substantially reduced and format violations are approximately 0%. The ablation study shows that more precise reformulation of instructions provides a slightly higher individual gain than examples alone, while their combination yields the best overall result at a moderate increase in risk. In this way, DSPy turns prompt design into a reproducible, measurable and cost-aware process, which is particularly important for safety-critical production systems based on large language models.

KEY WORDS: large language models, prompt engineering, DSPy, prompt optimization, malicious prompt classification.

1. УВОД

Развој великих језичких модела радикално је унапредио решавање задатака обраде природног језика (енг. *Natural Language Processing* – NLP) [1, 2]. Ипак, коначан квалитет одговора пресудно зависи од упита којим се моделу задаје задатак. Ручно обликовање упита је споро, захтева значајно искуство и често доводи до нестабилних исхода: и мале језичке измене могу да произведу велике разлике у резултатима. Поред тога, људи нису у стању да у упиту експлицитно изразе све обрасце и правила која интуитивно примењују, нити да обухвате све релевантне варијације ситуација, па се ручни *prompt* инжењеринг у пракси често показује као трошковно неефикасан и ограничен.

Овај проблем је посебно изражен у безбедносно осетљивим задацима, као што је класификација злонамерних упита (*jailbreak*), где је потребно истовремено контролисати тачност, ризик од пропуштања напада и оперативну стабилност система. Већина постојећих радова о оптимизацији упита усмерена је пре свега на повећање тачности или макро-F_1 мере на стандардним задацима обраде природног језика, док се ризик, нарушавање формата излаза и ограничен буџет позивања модела ретко третирају као равноправни критеријуми [4, 5]. На тај начин остаје истраживачки јаз између теоријски занимљивих метода оптимизације и практичних захтева продукционих, безбедносно оријентисаних система заснованих на великим језичким моделима (енг. *Large Language Models* – LLM).

Због ових изазова развијају се приступи који аутоматски оптимизују упите, третирајући их као параметре који се могу мењати тако да се максимизује изабрана метрика учинка [4, 5]. Уместо статичног, ручно писаног шаблона

текста, упит постаје објекат над којим се спроводи претрага или учење, на основу повратних резултата на валидационом скупу. Примери таквих метода су *AutoPrompt* и *Automatic Prompt Engineer*, који формулишу дизајн упита као проблем претраге у „црној кутији“, при чему се кандидати упита генеришу и евалуирају, а затим бирају боље варијанте [4, 5]. Детаљнији преглед ових приступа дат је у поглављу 2.

У овом раду усваја се DSPy оквир као репрезентативан пример програмског приступа оптимизацији упита [6]. DSPy омогућава да се систем заснован на LLM-у опише као програм с декларативно дефинисаним модулима и да се над таквим програмом покрену оптимизатори који аутоматски прилагођавају упите, контекст и примере. Структура DSPy програма коришћеног у овом раду, као и улога појединачних компоненти, детаљно су описани у поглављу 3.

Полазећи од ових увида, у раду се испитује у којој мери DSPy може аутоматски да научи ефикасније упите од ручно дефинисаних у задацима бинарне класификације злонамерних упита. Конкретно, примењује се MIPROv2 оптимизатор [8] (уз варијанте SIMBA и COPRO [6]) над моделом GPT-4.1-nano [9] на скупу података „jailbreak-classification“ [7], при чему се систематски контролишу три типа контекста: празан контекст (енг. *empty*), неистинит контекст (енг. *misleading*) и истинит контекст (енг. *true*), као и две полазне конфигурације упита (добар и лош почетни упит). Прате се не само тачност и макро-F_1, већ и ризик дефинисан као асиметрична казна за пропуштање злонамерних упита, као и стопа нарушавања формата излаза.

Главни доприноси рада су следећи. Прво, уводи се методолошки оквир за евалуацију оптимизације упита који комбинује учинак класификације, ризик и оперативну употребљивост (стопу нарушавања формата), чиме се нагласак помера са „само тачности“ на безбедност и поуздан излаз. Друго, операционализује се идеја „оптимизатора као компајлера“ кроз систематску примену MIPROv2 и његових варијанти под контролисаним условима, што омогућава фер поређење са ручно дефинисаним упитима и DSPy програмом без оптимизације. Треће, квантитативно се анализира улога контекста и преформулисања упита (енг. *rewrite*) кроз аблациону студију, која показује да информисан контекст делује као снажно предзнање (енг. *prior*) које убрзава и стабилизује оптимизацију, док прецизно преформулисање упита даје највећи појединачни допринос.

Остатак рада организован је на следећи начин. У поглављу 2 дају се теоријске основе: преглед великих језичких модела, prompt инжењеринга и приступа аутоматској оптимизацији упита. Поглавље 3 представља DSPy оквир и описује структуру програма коришћеног у експериментима. Поглавље 4 приказује експерименталну поставку и коришћени скуп података, док су у поглављу 5 изложени резултати и дискусија. На крају, поглавље 6 садржи закључак и правце даљег рада.

2. ТЕОРИЈСКА ОСНОВА

Савремени LLM модели, као што су GPT-3.5, LLaMA-2 и Gemma [1, 2, 13], заснивају се на Transformer архитектури [10] и обучавају се на великим корпусима текста, уз циљ да се науче општа правила природног језика [1, 2]. Поред тога, новији модели се често додатно фино подешавају (енг. *fine-tuning*) помоћу повратне информације од људи (енг. *reinforcement learning from human feedback* – RLHF), како би боље следили упутства и понашали се корисно и безопасно [11, 12]. Такви модели омогућавају решавање широког спектра задатака - од превођења до класификације, генерисања кода и дијалошких система - без класичног финог подешавања модела за сваки појединачни задатак. Уместо тога, понашање модела у великој мери одређује упит, односно текстуални опис задатка и контекста који корисник уноси. У режиму рада оријентисаном на упит, дизајн упита постаје кључна компонента.

Ручни *prompt* инжењеринг обухвата избор формулације задатка, ниво детаља у упиту, редослед и стил инструкција, као и могуће додавање неколико примера (енг. *few-shot*) који моделу илуструју жељено понашање [3]. Искуство показује да се учинак модела може значајно променити већ малим изменама у формулацији упита, редоследу наведених правила, избору термина или примерима који се дају [3]. Ипак, ове стратегије ослањају се на интуицију и експерименте стручњака, па су временски захтевне, субјективне и тешко репродуктивне. Посебан изазов јавља се у продукционим окружењима, где је потребно истовремено контролисати више фактора: тачност, стабилност излаза, поштовање формата (нпр. строга JSON или CSV структура), трошкове позивања модела и ризик од недозвољених или небезбедних одговора. Ручни *prompt* инжењеринг тешко може систематски да балансира све ове захтеве, што је подстакло развој формалнијих, аутоматизованих приступа дизајну упита.

Аутоматска оптимизација упита полази од идеје да упит не мора бити фиксан, ручно писан текст, већ објекат који се може представити параметрима и оптимизовати на основу мерљиве метрике учинка [4]. Уместо да човек проба велики број варијанти упита, систем организовано претражује простор могућих формулација – било кроз претрагу у дискретном простору текстуалних измена, било кроз учење континуалних представљања упита. Представници овог приступа су, између осталог, методе попут *AutoPrompt* и *Automatic Prompt Engineer*, које моделују формулисање упита као проблем претраге у „црној кутији“. У сваком кораку формира се скуп кандидатских упита, мери им се учинак на валидационом скупу и затим задржавају варијанте које постижу боље резултате [5].

DSPy оквир се надовезује на ову идеју, али иде корак даље у правцу програмског моделовања целог система заснованог на LLM-у [6]. Уместо да се рад са моделом своди на један текстуални упит, DSPy уводи појам `Signature`, декларативне спецификације улаза и излаза модула, и модула као програмске јединице која инкапсулира позив мо-

дела (нпр. `dspy.Predict` за предвиђање ознаке). Упит, инструкције и контекст за тај модул постају параметри које није потребно ручно фиксирати. Оквир омогућава да се они третирају као промењљиве које је могуће учити.

Кључни елемент DSPy-а јесу оптимизатори упита, који на основу валидационог скупа систематски побољшавају инструкције и примере тако да максимизују изабрану метрику (нпр. тачност, макро-F_1, поштовање формата излаза или комбинацију више циљева). MIPROv2, као један од централних алгоритама у оквиру DSPy-а, спроводи вишестепену оптимизацију у којој се истовремено разматрају преформулисање упита и избор кратких примера [8]. Уместо да корисник једнократно формулише упит, систем више пута „предлаже“ нове верзије, тестира их на валидационом скупу и задржава оне које доводе до бољег учинка.

Поред MIPROv2, развијене су и варијанте као што су SIMBA и COPRO [6], које се разликују по акценту између различитих компоненти упита (нпр. између правила и примера) и по томе како користе повратну информацију модела (саморефлексију, експлицитна правила, координатна побољшања). Ови алгоритми се могу посматрати као различити начини да се упит „разложи“ на делове и да сваки од њих пролази кроз цикличну, контролисану измену.

DSPy се, међутим, не ограничава само на једноставне класификационе модуле. Пошто су `Signature`-и и модули композициони, могуће је описати сложеније токове - ланце модула, изборне гране, кораке који прво из текста издвајају релевантне делове, а затим на њима раде класификацију или извлачење информација. Истим принципом, упит сваког модула може бити предмет оптимизације, а целина се посматра као програм који се „компајлира“ ка бољем понашању на конкретном задатку.

У пракси, овакви оквири су посебно релевантни у доменима где је комбинација фактора кључна: висока тачност, ниска стопа критичних грешака (нпр. лажно негативних у детекцији злоупотреба), поуздан формат излаза и ограничен трошак позивања модела. Уместо једнократног ручног дизајна, инжењер добија методологију у којој се:

- формално дефинише задатак (кроз `Signature`),
- опишу метрике и ограничења,
- и над тим оквиром покрећу алгоритми за аутоматску оптимизацију упита.

У наставку рада ова теоријска полазишта се конкретизују кроз примену DSPy програма заснованог на модулу `dspy.Predict` и оптимизаторима упита на задатку класификације злонамерних упита, уз анализу добитака у учинку и ризику у односу на ручно дефинисане и не-оптимизоване упите.

3. СКУП ПОДАТАКА И МЕТОДОЛОГИЈА

3.1. *СКУП „jailbreak-classification“*

За експерименталну анализу у овом раду коришћен је јавни скуп података „jailbreak-classification“, доступан на платформи *HuggingFace Datasets* [7]. Скуп је оригинално конструисан у сврху истраживања безбедности великих језичких модела, са циљем да омогући бинарну класификацију упита на:

- BENIGN - безбедне, уобичајене упите који не нарушавају безбедносне смернице,
- JAILBREAK - упите који покушавају да наведу модел да прекрши постављена правила или генерише недозвољен садржај.

Скуп садржи укупно 1306 текстуалних упита, при чему је приближно 50,5% примера означено као JAILBREAK. Оваква структура чини скуп релативно уравнотеженим и погодним за бинарну класификацију, без екстремне неравнотеже између класа. Сваки запис садржи два основна поља:

- prompt - текстуални упит који се шаље LLM-у,
- type - класа (BENIGN или JAILBREAK).

Сви упити су на енглеском језику и потичу из више извора: јавних примера техника злонамерних упита (нпр. „DAN“, „Evil Confidant“, „Do Anything Now“) и неутралних, безбедних упита преузетих из отворених корпуса (Alpaca, OpenAssistant и др.). Оваква комбинација злонамерних и неутралних упита омогућава да се у контролисаним условима испита робусност LLM-ова на различите типове напада и да се квантификује њихова осетљивост на стратегије злонамерних упита.

Ипак, постоји неколико ограничења која треба имати у виду. Обим скупа (1306 примера) је довољан за *proof-of-concept* анализу и контролисане експерименте, али није довољан за обуку сложенијих неуронских архитектура које захтевају много веће количине података. Поред тога, већина упита потиче из ограниченог броја извора и сви су на енглеском језику, што ограничава директну применљивост резултата на вишејезичне сценарије и на потпуно нове стилове злонамерних упита. У овом раду скуп се зато користи као референтно полазиште за процену добитака које доноси аутоматска оптимизација упита, а не као коначни представник свих могућих напада.

3.2. *ЕКСПЕРИМЕНТАЛНО ОКРУЖЕЊЕ*

Експерименти су извођени у Python 3.11 окружењу, у оквиру изолованог виртуелног окружења и са фиксним верзијама кључних библиотека (`dspy-ai`, `openai`, `datasets`, `pandas`, `numpy`, `scikit-learn`, `python-dotenv`). На тај начин обезбеђена је репродуктивност резултата и избегнуте су нежељене разлике услед промена верзија пакета.

Скрипте су покретане локално, док се само извршавање LLM-а одвијало на страни *cloud* провајдера (OpenAI API). Локална машина је служила за оркестрацију експеримената, управљање подацима, бележење логова и израчунавање метрика, без потребе за локалним GPU-ом. Основне карактеристике окружења (архитектура x86_64, најмање 8 GB RAM, операциони систем Windows 10) биле су стабилне током целог истраживања.

Сви конфигурациони параметри (тип модела, температура, максималан број токена, лимит трошкова и др.) чувани су у посебној `.env` датотеци, која није део верзионисаног кода. При покретању скрипте помоћна функција учитава ове вредности, поставља случајни `SEED` и формира конфигурациони објекат који се прослеђује остатку система. На тај начин је:

- обезбеђена репродуктивност (фиксиране случајне почетне вредности у `random`, `NumPy` и DSPy-у),
- омогућено лако прилагођавање различитим провајдерима и моделима без измене основне логике, и
- обезбеђено сигурно руковање тајнама (API кључеви нису експлицитно присутни у коду).

У свим експериментима коришћен је модел GPT-4.1-nano као компромис између квалитета и трошкова. Параметар `temperature` је држан у опсегу 0,0-0,2, како би се минимизовала стохастичка варијабилност и омогућило да разлике у учинку што више одражавају квалитет упита, а не случајне флуктуације у одговорима модела. Максимална дужина одговора ограничена је на разумну вредност тако да се испоштује формат излаза и контролише број токена по позиву.

Скуп „jailbreak-classification“ је, након учитавања, подељен на тренинг, валидациони и тест део, уз стратификовано распоређивање примера и фиксни `random_state`. Оваква организација обезбеђује уједначену дистрибуцију класа у свим подскуповима и спречава цурење информација из валидационог у тестни део, што је посебно важно јер се оптимизација упита врши управо на валидационом скупу.

Да би се избегли непотребни трошкови API позива, уведен је локални кеш (енг. *cache*). Резултати одговора се хеширају на основу комбинације модела, температуре, системског упита и корисничког текста. У случају поновљеног позива са истом конфигурацијом, користи се раније кеширани резултат уместо новог упита ка моделу. Поред тога, примењен је експоненцијални *backoff* у случају грешака или ограничења брзине (енг. *rate limit*), тако да систем аутоматски понавља покушај након пауза.

3.3. *DSPy ПРОГРАМ И ЕКСПЕРИМЕНТАЛНА ПОСТАВКА*

Методолошко језгро овог рада чини DSPy програм заснован на модулу `dspy.Predict`, који LLM-у задаје задатак бинарне класификације упита (BENIGN / JAILBREAK). Уместо класичног „ручног“ упита, модел се описује преко:

- `Signature`-а који декларативно дефинише улаз (текст упита) и излаз (класа или структурисани одговор),
- и модула који инкапсулира позив LLM-у, као и пратеће инструкције, контекст и примере.

У овом истраживању `Signature` спецификује да модул прима један текстуални упит и враћа ознаку класе. Сам текст инструкција (објашњење задатка, правила понашања, евентуална упутства о формату) и избор примера који илуструју типичне BENIGN и JAILBREAK случајеве третирају се као параметри које DSPy може да мења и оптимизује.

Експериментална поставка обухвата две групе конфигурација:

- базна линија, у којој се користи ручно дефинисан упит (једна добра и једна намерно лоша формулација), уз различите поставке контекста (истинит, неистинит, празан),
- DSPy конфигурација, у којој се исти задатак формулише као DSPy програм, а инструкције и примери се оптимизују помоћу уграђених оптимизатора.

Главни оптимизатор коришћен у раду је MIPROv2, који комбинује *rewrite* и *few-shot*. Поред њега, разматране су и варијанте SIMBA и COPRO, али је фокус на анализи добитака које доноси MIPROv2 у односу на:

- ручно дефинисан упит,
- и DSPy програм без фазе оптимизације (фиксни модул, без измена инструкција и примера).

Током оптимизације, алгоритам више пута генерише кандидатске верзије упита, тестира их на валидационом скупу и задржава оне које унапређују изабрану метрику. Као циљна функција користи се пре свега макро-F_1, а у проширеној анализи и:

- тачност (енг. *accuracy*),
- ризик, дефинисан асиметричном ценом грешака (већа казна за лажно негативне (FN) у случају JAILBREAK примера. Тачније, ризик је дефинисан као збир пет пута лажно негативни и један пута лажно позитивни),
- и стопа нарушавања формата, која мери колико често модел не поштује очекивану структуру излаза.

Сви експерименти се изводе под ограниченим буџетом - укупни трошак позива модела се прати и не сме да пређе задату вредност. На тај начин, методологија симулира реалне продукционе услове у којима је потребно балансирати између квалитета, стабилности и цене.

Детаљни резултати поређења базних линија, DSPy програма без оптимизације и MIPROv2 оптимизованих конфигурација приказани су у наредном поглављу, уз анализу ефекта појединачних компоненти (*rewrite*, *few-shot*, промена контекста) на укупан учинак и ризик.

4. РЕЗУЛТАТИ И ДИСКУСИЈА

4.1. КВАНТИТАТИВНИ *РЕЗУЛТАТИ*

Евалуација је спроведена на фиксној, стратификованој подели скупа „jailbreak-classification“ на тренинг, валидациони и тест део (70/15/15), уз исти `SEED` и стабилну кон-

фигурацију модела. Све варијанте су рађене у IID условима. Оптимизација упита вршена је на валидационом скупу, док су пријављени бројеви добијени на издвојеном тестном скупу. Примарна метрика је макро-F_1, уз пратеће мере тачности, ризика и стопе нарушавања формата. У табели 1 су упоредно приказани резултати за исти модел и исту поделу података, при варирању контекста и оптимизационог режима. У *baseline* редовима почетни системски упит је фиксан и намерно лош; у MiPROv2 редовима упит је аутоматски унапређен. У DSPy рундама стопа нарушавања формата је ≈ 0%, што омогућава поуздано укључивање у даље експерименте.

У програмском коду контекст је параметар који може да има вредности *empty*, *misleading* и *true*, па су исте енглеске ознаке коришћене и у табели. У наставку текста користе се српски називи: празан контекст (*empty*), неистинит контекст (*misleading*) и истинит контекст (*true*).

Табела 1. *Резултати експеримената*

Поставка Скрипта Оптимизатор Контекст	*Rewrite*	Вал. - # FN	Вал. – # FP	Вал. – Ризик	Вал. – Нарушавање формата	Тест – Тачност	Тест – макро-F_1	Вал. – макро-F_1
Baseline run_baseline.py / empty	НЕ	/	/	/	0,64%	0,318	0,248	/
DSPy (opt=none) dspy_version_param_ prompt.py none empty	НЕ	23	45	160	0,00%	0,701	0,689	0,556
DSPy (opt=none) dspy_version_param_ prompt.py none misleading	НЕ	10	44	94	0,00%	0,682	0,650	0,636
MiPROv2 dspy_version_param_ prompt.py miprov2 empty	ДА	12	17	77	0,00%	0,878	0,878	0,866
MiPROv2 dspy_version_param_ prompt.py miprov2 misleading	ДА	6	23	53	0,00%	0,860	0,857	0,812
MiPROv2 dspy_version_param_ prompt.py miprov2 true	ДА	15	11	86	0,00%	0,936	0,936	0,834
Ceiling (good prompt) dspy_version_param_ prompt.py / true	/	/	/	/	/	0,955	0,955	0,955

Као полазна тачка узета је базна линија без DSPy оквира, са намерно лошим почетним упитом и празним контекстом. У тој поставци модел постиже макро-F_1 ≈ 0,248, што показује да је осетљив и на квалитет упита и на недостатак доменских смерница. Увођење DSPy програма без фазе оптимизације (фиксни модул, без MIPROv2/SIMBA/COPRO) већ доноси значајан скок. За исту поделу података, исти модел и празан контекст макро-F_1 расте на ≈ 0,689. Ово показује да и само прелазак на декларативни опис задатка и бољу организацију упита даје опипљив добитак.

Највећи напредак постиже се када се укључи MIPROv2 оптимизатор. За празан контекст MIPROv2 подиже макро-F_1 на ≈ 0,878, уз пад ризика у односу на варијанту без оптимизације. Када се, поред тога, уведе и истинит контекст, макро-F_1 достиже ≈ 0,936 и приближава се референтној „ceiling“ вредности ≈ 0,955 добијеној са добрим почетним упитом. У свим DSPy рундама стопа нарушавања формата је занемарљива (≈ 0%), што је практично важно за даље коришћење излаза у продукционим системима.

Посматрано у целини, прелазак са ручно дефинисаног упита на DSPy + MIPROv2 + истинит контекст даје скок са макро-F_1 ≈ 0,248 на приближно 0,936, уз истовремено смањење ризика и стабилан формат излаза. То показује да аутоматска оптимизација упита уз контролисан контекст може да надомести велики део ручног *prompt* инжењеринга.

4.2. *ЕФЕКАТ КОНТЕКСТА КАО PRIOR-А И ПРЕФОРМУЛАЦИЈЕ УПИТА*

Експерименти са различитим контекстима (истинит, неистинит, празан) омогућавају да се квантитативно процени улога доменских смерница. Са празним контекстом, MIPROv2 већ значајно побољшава учинак у односу на DSPy конфигурацију без оптимизације (макро-F_1 ≈ 0,878 наспрам ≈ 0,689), уз пад ризика (мање лажно негативних и укупно „скупих“ грешака). То указује да сам *rewrite* систематски исправља неадекватни почетни упит и стабилизује одлуке.

Додавање истинитог контекста додатно подиже учинак: са ≈ 0,878 (MIPROv2 и празан контекст) на ≈ 0,936 (MIPROv2 и истинит контекст), што представља релативни пораст од око 6 – 7%. У односу на базну DSPy варијанту без оптимизације (означену као *none* у табели 1 и празан контекст, макро-F_1 ≈ 0,689) укупан добитак износи око +0,247, односно више од трећине релативног побољшања. Ови резултати емпиријски потврђују тезу да добро дизајниран контекст делује као снажно предзнање (енг. *prior*): усмерава модел ка релевантним сигналима (нпр. шаблони заобилажења политика, социјални инжењеринг, обфускација) и пре самог преформулисања упита.

Посебно је интересантан случај неистинитог контекста, који намерно уводи погрешне или контрадикторне информације. Када се примени без оптимизације, овакве смернице деградирају учинак моделирајући лош *prior*. Међутим, комбинација MIPROv2 и неистинитог контекста и даље значајно надмашује базну линију. Макро-F_1 расте у односу на *none* из табеле 1 и празан контекст, а ризик се смањује. То показује да систематска оптимизација може делимично да ублажи штету лошег контекста, иако је комбинација MIPROv2 и истинит контекст и даље најбоља.

У целини, резултати указују да се *rewrite* и контекст допуњују. *Rewrite* стабилизује одлуке и поправља неадекватан почетни упит, док истинит контекст даје додатни добитак и приближава учинак ручно дизајнираној „ceiling“ конфигурацији.

4.3. *АБЛАЦИЈА: REWRITE У ОДНОСУ НА FEW-SHOT ПРИМЕРЕ*

Да би се изоловао допринос појединачних компоненти MIPROv2 оптимизације, спроведена је аблација са три варијанте при фиксном истинитом контексту:

- *rewrite-only* - активна је само преформулација упита, без демонстрационих примера,
- *demos-only* - избор кратких примера, без преписа инструкција,
- *full* MiPROv2 - истовремено активни и преформулација упита и избор примера.

Све три варијанте рађене су под истим условима (исти модел, иста подела, исти `SEED`, исти „интензитет претраге“).

Резултати на тест скупу показују:

Табела 2. *Резултати након аблација*

Варијанта (аблација)	Тест – макро-F_1	Тест - Тачност	Вал. – марко-F_1	Вал. – Ризик
MiPROv2 - rewrite-only	**0,9043**	0,9045	0,8468	**64**
MiPROv2 - demos-only	**0,8848**	0,8854	0,8599	**70**
MiPROv2 - full	**0,9362**	0,9363	0,8344	**86**

Поређење *rewrite-only* и *demos-only* указује да је *rewrite* јачи појединачни допринос: макро-F_1 је виши за ≈ 0,02, а ризик је нижи (64 наспрам 70). То значи да јасно, доследно формулисање задатка и правила (уз нагласак да излаз буде једна ознака) већ само по себи доноси значајно побољшање профила одлука, чак и без иједног примера.

Комбинована варијанта (*rewrite* + *few-shot*) даје највиши макро-F_1 ≈ 0,936, што је за ≈ 0,032 више од чистог *rewrite*-а и ≈ 0,051 више од чистих примера. Просек појединачних варијанти је ≈ 0,895, док пуна варијанта достиже ≈ 0,936, што указује на синергију: *rewrite* и примери нису замена једно за друго, већ се међусобно допуњују и у комбинацији дају више него што би се очекивало из њихових засебних ефеката.

С друге стране, комбиновна варијанта има нешто виши ризик (≈ 86) у односу на *rewrite-only* (64) и *demos-only* (70). То значи да агресивнија оптимизација понекад „оштри“ граничну површину у корист F_1, по цену нешто већег ризика у односу на задату ценовну функцију (у овом раду лажно негативне грешке су вишеструко скупље). У практичној примени, однос између пропуштених опасних примера (FN) и лажних узбуна (FP) може се контролисати на више начина: избором циљне метрике у оптимизацији (нпр. пондерисани F_1 или ризик), увођењем правила која током претраге одбацују кандидате са превише FN, или накнадном калибрацијом одлука.

4.4. *ПРАКТИЧНЕ ИМПЛИКАЦИЈЕ*

Резултати показују да DSPy + MIPROv2 може, у реалистичним условима ограниченог буџета и строгог формата излаза, да практично замени скупо ручно подешавање упита: најбоља конфигурација (MIPROv2 и истинит контекст али са лошим почетним упитом) достиже макро-F_1 ≈ 0,936, врло близу ручно дизајнираног „ceiling“ упита-а (≈ 0,955). Истовремено, стопа нарушавања формата остаје ≈ 0%, што је критично за све сценарије у којима се одговори аутоматски даље обрађују.

Аблациона анализа сугерише јасан приоритет за инжењерску праксу:

- као први корак, вреди систематски уредити и преформулисати упит,
- затим додати мали број пажљиво бираних примера,
- и на крају, увести добро осмишљен контекст који моделу даје стабилан *prior*.

Ова методологија третира упит као параметар који се учи, а не као статичан текстуални шаблон. У доменима као што је детекција злонамерних упита то омогућава да се уз разумне трошкове постигну висока тачност, низак ризик и стабилан формат излаза, уз јасну процедуру за даље унапређење по потреби.

5. ЗАКЉУЧАК И ПРАВЦИ ДАЉЕГ РАДА

Овај рад је полазио од проблема осетљивости великих језичких модела на формулацију упита и ограничења ручног *prompt* инжењеринга, посебно у безбедносно осетљивим задацима као што је класификација злонамерних упита. Уместо статичног, ручно дефинисаног шаблона текста, предложен је приступ у ком се упит третира као параметар који је могуће систематски оптимизовати, уз јасно дефинисане метрике учинка и ризика. У том контексту примењен је DSPy оквир, који омогућава да се систем заснован на LLM-у опише као програм с декларативним `Signature`-има и модулима, а затим „компајлира“ тако да максимизује изабрану метрику. То је посебно важно у контексту *foundation* модела и њиховог утицаја на реалне системе [13].

Експерименти на скупу „jailbreak-classification“ показали су да прелазак са ручно формулисаног упита на DSPy програм и MIPROv2 оптимизацију доводи до значајних добитака. Базна линија постиже макро-F_1 ≈ 0,248, док DSPy програм без оптимизације подиже учинак на ≈ 0,689. Укључивањем MIPROv2, при празном контексту, макро-F_1 расте до ≈ 0,878, а уз добро дизајниран контекст достиже ≈ 0,936, веома близу ручно осмишљене „ceiling“ конфигу-

рације. Истовремено, ризик опада, а нарушавање формата излаза је практично занемарљиво, што је од суштинског значаја за продукционе системе који аутоматски обрађују одговоре модела.

Аблациона анализа је показала да је преформулисање упита (*rewrite-only*) снажнији појединачни допринос од самих примера (*demos-only*), али да њихова комбинација даје најбоље резултате. Ово указује да је најделотворнија стратегија у пракси она која прво систематски уреди и прецизира инструкције, затим дода мали број пажљиво одабраних примера и на крају уводи доменски контекст као *prior*. На тај начин, DSPy и сродни оквири омогућавају да се дизајн упита формализује као процес учења и оптимизације, уместо да остане занат који се ослања на интуицију и ad-hoc експерименте.

Ипак, овај рад има и своја ограничења. Анализа је спроведена на једном, релативно малом скупу података, на једном моделу (GPT-4.1-nano) и у специфичном домену бинарне класификације злонамерних упита на енглеском језику. Није разматрано понашање већих модела, вишејезични сценарији, други типови задатака (нпр. извлачење ентитета, кластеровање или сложени токови одлучивања), нити дугорочно одржавање оптимизованих упута у условима промене напада и политика. Поред тога, циљна функција је доминантно била макро-F_1, док би у појединим апликацијама директна оптимизација ризика или комбинација више метрика могла бити релевантнија.

Правци даљег рада обухватају проширење експеримената на веће и разноврсније скупове података, укључивање више модела и језика, као и испитивање утицаја различитих дефиниција ризика на ток оптимизације. Посебно је интересантно систематско поређење DSPy приступа са другим оквирима за аутоматску оптимизацију упита, као и интеграција оптимизованих упита у реалне безбедносне *pipeline*-ове, где би се њихов учинак пратио у дугом временском периоду. На тај начин, резултати овог рада могу послужити као основа за развој методологија у којима се упит посматра као динамичан, мерљив и управљив ресурс у дизајну система заснованих на LLM-овима, а не као непроменљив текстуални шаблон.

ЛИТЕРАТУРА

[1] Brown, T. B., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33.

[2] Hoffmann, J., et al. (2022). Training compute-optimal large language models. Advances in Neural Information Processing Systems, 35.

[3] Lu, Y., et al. (2022). Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. Proceedings of ACL 2022.

[4] Shin, T., Razeghi, Y., Logan IV, R. L., Wallace, E., & Singh, S. (2020). AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. Proceedings of EMNLP 2020.

[5] Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2022, November). Large language models are human-level prompt engineers. Proceedings of ICLR 2023.

[6] Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., ... & Potts, C. (2023). Dspy: Compiling declarative language model calls into self-improving pipelines.

[7] jackhhao (2025). jailbreak-classification [Dataset]. Hugging Face. [https://huggingface.co/datasets/jackhhao/jailbreak-classification]

[8] Opsahl-Ong, K., Ryan, M. J., Purtell, J., Broman, D., Potts, C., Zaharia, M., & Khattab, O. (2024). Optimizing instructions and demonstrations for multi-stage language model programs. Proceedings of EMNLP 2024.

[9] OpenAI. (2025). GPT-4.1-nano [Large language model]. OpenAI Platform Documentation. [https://platform.openai.com/docs/models/gpt-4.1-nano]

[10] Vaswani, A., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

[11] Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35.

[12] Bai, Y., et al. (2022). Training a helpful and harmless assistant with reinforcement learning from human feedback. [https://arxiv.org/abs/2204.05862]

[13] Bommasani, R., et al. (2021). On the opportunities and risks of foundation models. [https://arxiv.org/abs/2108.07258]

Тара Милојковић, компанија Rivian and Volkswagen Group Technologies на позицији Data Engineering Intern - Analytics & AI

Контакт: taramilojkovic6@gmail.com

Област интересовања: Data Engineering, Large Language Models (LLMs)

UPUTSTVO ZA PRIPREMU RADA

1. Tekst pripremiti kao Word dokument, A4, u kodnom rasporedu 1250 latinica ili 1251 ćirilica, na srpskom jeziku, bez slika. Preporučen obim – oko 10 strana, single prored, font 11.
2. Naslov, abstakt (100-250 reči) i ključne reči (3-10) dati na srpskom i engleskom jeziku.
3. Jedino formatiranje teksta je normal, bold, italic i bolditalic, VELIKA i mala slova (tekst se naknadno prelama).
4. Mesta gde treba ubaciti slike, naglasiti u tekstu (Slika1...)
5. Slike pripremiti odvojeno, VAN teksta, imenovati ih kao u tekstu, radi identifikacije, u sledećim formatima: rasterske slike: jpg, tif, psd, u rezoluciji 300 dpi 1:1 (fotografije, ekranski prikazi i sl.), vektorske slike – cdr, ai, fh,eps (šeme i grafikoni).
6. Autor(i) treba da obavezno priloži svoju fotografiju (jpg oko 50 Kb), navede instituciju u kojoj radi, kontakt i 2-4 oblasti kojima se bavi.
7. Maksimalni broj autora po jednom radu je 5.

Redakcija časopisa Info M