VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

AUTOMATICKÁ TVORBA ANIMOVANÉHO VIDEA NA ZÁKLADĚ TEXTOVÉHO PŘÍBĚHU

AUTOMATIC CREATION OF ANIMATED VIDEO BASED ON TEXTUAL STORY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JOSEF KUCHAŘ**
AUTHOR

VEDOUCÍ PRÁCE — **doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

155488

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Kuchař Josef**
Program:	Informační technologie
Název:	**Automatická tvorba animovaného videa na základě textového příběhu**
Kategorie:	Počítačová grafika
Akademický rok:	2023/24

Zadání:

1. Seznamte se s moderními metodami generování animací na základě difúzních modelů, procedurální animace
2. Zpracujte přehled dostupných předučených modelů a technik pro udržení konzistence postav a prostředí, nutné pro animaci příběhů
3. Na základě získaných poznatků navrhněte a implementujte systém, který dokáže s definovanými omezeními generovat konzistentní animaci příběhů
4. Vyhodnoťte výsledky systému v uživatelské studii, zaměřené na konzistenci ztvárnění postav a struktury vyprávění a intuitivnost ovládání.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího, mj.:
- https://huggingface.co/nitrosocke/classic-anim-diffusion
- https://the-decoder.com/motion-diffusion-turns-text-into-lifelike-human-animations/

Při obhajobě semestrální části projektu je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	21.12.2023

Abstrakt

Cílem této práce je propojit difúzní model pro generování lidského pohybu s difúzním modelem generující video. V řešení jsou použité aktuální metody pro generování videa a pohybu. Generování videa probíhá pomocí generátoru obrázků opatřeného adaptérem pro temporální konzistenci. Práce představuje metodu propojení obou difúzních modelů pomocí sítě ControlNet. Vytvořené řešení umožňuje generovat video z jednoduchého textového popisu, případně detailního scénáře. Program byl otestován v uživatelské studii.

Abstract

The aim of this work is to link a diffusion model for generating human motion with a diffusion model for generating video. The solution uses current methods for generating video and motion. Video generation is carried out using an image generator equipped with an adapter for temporal consistency. The work introduces a method of connecting both diffusion models using the ControlNet network. The created solution allows for generating video from a simple text description, or a detailed scenario. The program was tested in a user study.

Klíčová slova

neuronové sítě, umělá inteligence, difúzní modely, generování videa, textový příběh, procedurální animace, ControlNet, Stable Diffusion, MDM, Human Motion Diffusion Model

Keywords

neural networks, artificial intelligence, diffusion models, video generation, textual story, procedural animation, ControlNet, Stable Diffusion, MDM, Human Motion Diffusion Model

Citace

KUCHAŘ, Josef. *Automatická tvorba animovaného videa na základě textového příběhu.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Pavel Smrž, Ph.D.

Automatická tvorba animovaného videa na základě textového příběhu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Josef Kuchař
6. května 2024

Poděkování

Rád bych poděkoval doc. RNDr. Pavlu Smržovi, Ph.D. za jeho vstřícnost a trpělivé vedení mé práce a rodičům za podporu během celé doby studia. Děkuji také všem, kteří se zúčastnili uživatelské studie.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V posledních letech se umělá inteligence stává klíčovým nástrojem v oblasti počítačového vidění a multimediálního obsahu, což vedlo k vývoji pokročilých metod pro generování realistických obrázků. Postupně se tato oblast přesouvá ke generování animací a videí. Jednou z nejnovějších a nejslibnějších technologií v této oblasti jsou difúzní modely, které umožňují generování obsahu na základě textového popisu, případně dalších jiných vstupů, jakými jsou například obrázky. Přestože samostatné difúzní modely pro generování videí a pro modelování lidského pohybu již existují, spojení těchto dvou řešení do jednotného systému pro generování animací s plynulým a diverzním lidským pohybem zůstává významnou výzvou.

Cílem mé bakalářské práce je překonat tuto mezeru prostřednictvím vývoje systému, který propojuje difúzní model pro generování lidského pohybu s difúzním modelem pro generování videa. Klíčem k dosažení tohoto cíle je využití nejnovějších postupů a technik v oblasti generování videa a pohybu. Kombinací generátoru obrázků opatřeného adaptérem AnimateDiff pro zajištění temporální konzistence a vývojem nové metody pro propojení těchto modelů pomocí neuronové sítě ControlNet se tato práce snaží otevřít nové možnosti pro tvorbu animací z jednoduchých textových popisů, nebo detailnějších scénářů. Scénář je v případě použití jednoduchého textového popisu příběhu vygenerován pomocí velkého jazykového modelu podobného tomu, který můžeme znát ze služeb jako je OpenAI ChatGPT, nebo Google Gemini.

Přínos této práce spočívá v demonstraci schopnosti generovat animace, které jsou založené na plynulém a přirozeném lidském pohybu. Důraz je také kladen na vizuální konzistenci postav a prostředí ve kterých se nacházejí. Aby bylo možné ověřit funkčnost navrženého systému, bylo provedeno testování v rámci uživatelské studie.

V následujících kapitolách je nejprve představen přehled současných metod a technologií využívaných v oblasti generování obrázků, videí a lidského pohybu. Následně je podrobně popsán návrh a implementace systému, včetně popisu jednotlivých komponent a jejich vzájemné interakce. Zvláštní pozornost je věnována metodice propojení difúzních modelů pomocí sítě ControlNet a analýze výsledků uživatelské studie, která hodnotí funkčnost a užitečnost vytvořeného řešení. V závěru práce je shrnutí celkových výsledků a nastínění možností pro další rozšíření.

Kapitola 2

Metody generování obsahu

V této kapitole budou nastíněny a vysvětleny metody pro generování obrázků, videa a lidského pohybu. Výčet metod je velký, proto do detailů budou rozebrány metody použité v této práci. Ostatní metody jsou zde uvedeny pro získání přehledu o aktuálním stavu výzkumu.

2.1 Difúzní modely

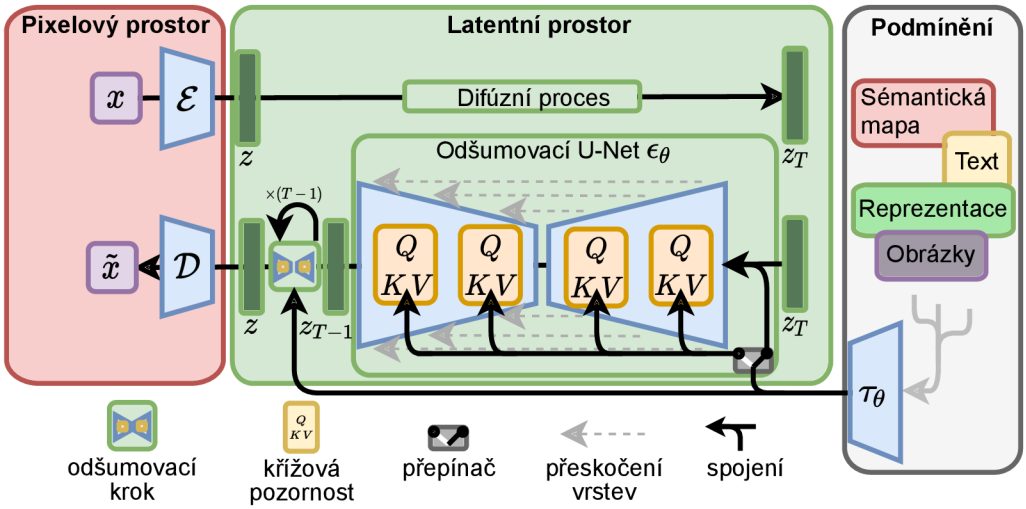
Difúzní modely pracují na principu iterativního odstraňování šumu. Architektura sítě pro odstraňování šumu může být různá. V oblasti generování obrázků a videí je populární architektura U-Net, ale postupně se pozornost přesouvá na architekturu transformer.

Latentní difúzní model

Latentní difúzní modely pro generování obrázků a videí mají 3 primární části:

- textový enkodér
- odšumovací síť
- variační autoenkodér

Textový enkodér nejprve převádí textový vstup (*prompt*) na vektorovou reprezentaci (*embedding*). Pomocí této reprezentace odšumovací síť iterativně odstraňuje šum z obrázku ve zmenšeném (*latentním*) prostoru a následně variační autoenkodér obrázek z latentního prostoru převádí do pixelového prostoru pro získání finálního obrázku, případně videa. Základní schéma latentního difúzního modelu Stable Diffusion je znázorněno na obrázku 2.1. Odšumovací proces je znázorněný na obrázku 2.2. Tato sekce vychází z článku [4].

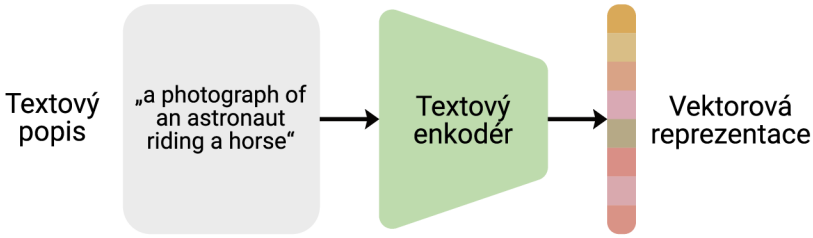


Obrázek 2.1: Schéma latentního difúzního modelu Stable Diffusion. Převzato z [19] (Upraveno).

Obrázek 2.2: Znázorněný proces odšumnění

2.1.1 Textový enkodér

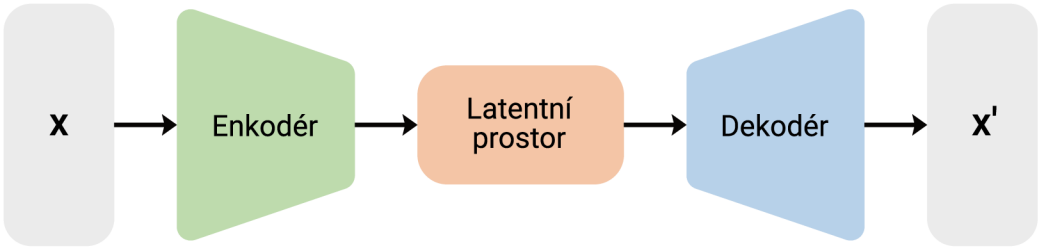
Textový enkodér v Stable Diffusion je předtrénovaný model CLIP[17], který využívá architekturu Vision Transformer. Tento enkodér převádí text na vektorovou reprezentaci (anglicky *embedding*). Latentní vektor zachycuje sémantický význam (případně by se také dalo říci esenci) vstupního textu a slouží jako podmíněný vstup, který řídí difúzní model během generování obrázku. Fungování textového enkodéru je zachyceno na obrázku 2.3.



Obrázek 2.3: Princip textového enkodéru

2.1.2 Variační autoenkodér

Variační autoenkodér slouží pro získání menší reprezentace obrázku. Jedná se vlastně o typ ztrátové komprese. Zakódovaná latentní reprezentace z variačního enkodéru je výrazně menší než původní obrázek, u Stable Diffusion 1.5 přibližně 48krát menší (například redukce obrázku $3 \times 512 \times 512$ na latentní $4 \times 64 \times 64$). Během dekódovacího procesu síť dekodéru zvětšuje tyto kompaktní latentní vektory zpět na původní rozměry obrázku pomocí série konvolučních a transponovaných konvolučních vrstev. Díky použití variačního enkodéru nemusí odšumovací prvek pracovat s tak obrovským prostorem, čímž se výrazně zjednodušuje trénování. Princip variačního autoenkodéru je zachycen na obrázku 2.4.



Obrázek 2.4: Schéma variačního autoenkodéru

2.1.3 U-Net

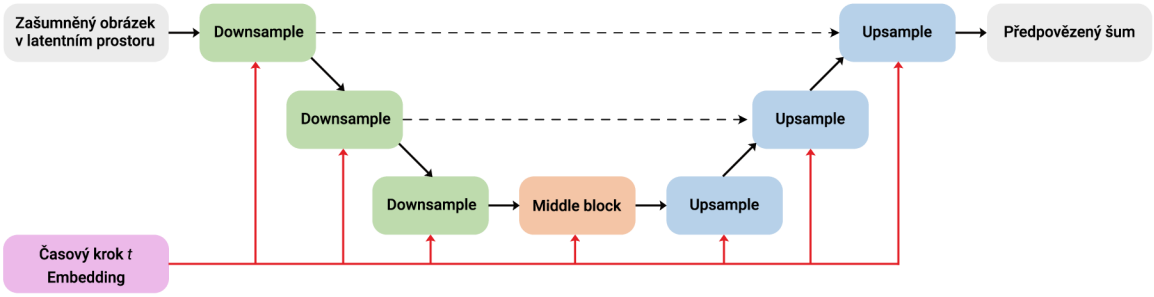
U-Net je konvoluční neuronová síť, která funguje jako odšumovací prvek v latentním difuzním modelu Stable Diffusion. Pracuje se dvěma vstupy:

- vektorovou reprezentací
- latentní reprezentací obrázku

Architektura U-Net je velmi podobná variačnímu enkodéru. Skládá se z enkodérové části, která používá konvoluce ke kompresi vstupu do feature map s nižším rozlišením. Dekodér

naopak z tohoto nižšího rozlišení rekonstruuje původní obrázek. Důležitým prvkem architektury jsou skip[1] spojení, která spojují výstupy enkodéru se vstupy dekodéru v odpovídajících rozlišeních. Tento design skip spojení pomáhá optimalizovat lokalizaci a tok informací.

U-Net produkuje předpověď reziduálního šumu pro každý časový krok. Odšumnění latentní reprezentace probíhá pomocí předpovězeného šumu a jeho zpětného zasílání do U-Net sítě postupně rekonstruuje konečný obrázek, který je řízen textovými vektory. Schéma sítě U-Net je na obrázku 2.5.



Obrázek 2.5: Zjednodušené schéma architektury U-Net

2.2 Přehled metod pro generování videa

V této sekci se nachází přehled metod pro generování videa pomocí difúzních modelů. Tento přehled je dělený na dvě kategorie. První kategorie jsou difúzní modely přímo určené pro generování videa. V druhé kategorii se nachází modely pro generování obrázků opatřené dodatečnou temporální konzistencí.

2.2.1 Generování videa pomocí difúzního video modelu

V této části bude představen nekompletní výčet aktuálních metod pro generování videa pomocí difúzního video modelu. Primárně jsou zde rozebrány metody a modely, které se dají lokálně spustit. Tyto modely byly autorem práce otestovány:

ModelScopeT2V[22] Architektura modelu ModelScopeT2V je jednou z nejstarších a je stále populární. Existuje několik předtrénovaných modelů o různých rozlišeních. Nejlepší model je `zeroscope_v2_XL`, který produkuje videa s rozlišením 1024×576 pixelů. Výhodou tohoto modelu je, že dokáže generovat teoreticky neomezeně dlouhá videa.

Tento model nebyl v této práci použit kvůli nemožnosti dále ho řídit například pomocí sítě ControlNet.

Stable Video Diffusion[2] Model Stable Video Diffusion generuje 14 (případně 25) snímků videa na základě vstupního počátečního obrázku. Výstupní video má rozlišení 1024×576 pixelů. K dosažení delšího videa lze použít poslední vygenerovaný snímek jako počáteční snímek pro generování.

Z mého testování funguje model skvěle na některých vstupních obrázcích, na některých zase vůbec (model vyprodukuje statické snímky). Prodlužování videa pomocí výše popsané

[1] https://www.analyticsvidhya.com/blog/2021/08/all-you-need-to-know-about-skip-connections/

techniky není příliš funkční kvůli absenci informací o předchozím pohybu. Model není možné řídit pomocí textu ani sítě ControlNet.

Show-1[26] Technika Show-1 využívá několika modelů v kaskádě. S každým krokem se postupně zvětšuje rozlišení videa, posledním krokem je vyše zmíněný ModelScopeT2V, pracující v režimu video-to-video. Výsledkem je video o rozlišení 576×320 pixelů a délkou 29 snímků. Vstupem do modelu je text.

Tuto techniku jsem rovněž otestoval. Dosahuje obstojných konzistentních výsledků, ale kvůli absenci možnosti prodlužování nemohla být použita.

I2VGenXL[29] Tato technika rovněž využívá modely v kaskádě. Generuje 16 snímků s rozlišením 1280×720 pixelů. Vstupem do modelu je řídící obrázek a text. To umožňuje lepší kontrolu nad vygenerovaným obsahem. Video lze prodloužit opět využitím posledního vygenerovaného snímku jako řídící snímek pro dalších 16 snímků.

Bohužel mé testování ukázalo, že výsledky nejsou konzistentní. U některých obrázků model vygeneruje skvělé koherentní video, u jiných se scéna kompletně rozpadne.

2.2.2 Generování videa s využitím generátorů obrázků

Tyto metody využívají existující difúzní generátory obrázků pro generování videa. Fungují na tom principu, že určitým způsobem vytvářejí konzistenci mezi jednotlivými obrázky pro dosažení temporální konzistence. Mají několik výhod. První výhodou jsou malé nároky na výpočetní výkon oproti trénování generátoru videa od základu. Druhou velkou výhodou je možnost využití existujících modelů a metod pro personalizaci modelů. Zároveň je možné využít metody pro řízení syntézy pomocí sítě ControlNet. Hlavní dvě metody jsou:

Text2Video-Zero[8] Metoda Text2Video-Zero využívá existující metody syntézy obrázku z textu pro adaptaci na video. Metoda není omezena pouze na syntézu videa z textu, ale je aplikovatelná i na další úkoly, jako je návodem řízená úprava videí. Zajímavostí techniky je fakt, že nevyužívá žádný další model, celá je dosažena pouze úpravou architektury existujícího difúzního modelu bez dalšího trénování. Tento popis vychází z abstraktu Text2Video-Zero[8]. Hlavní výhodou této techniky je možnost generování teoreticky neomezeně dlouhých videí.

Z mého testování je tato metoda funkční, avšak výsledky nejsou dostatečné pro tuto práci. Pozadí je skoro úplně statické, pohyb postavy zanechává velké vizuální artefakty. Další nevýhodou této metody je, že udržuje temporální konzistenci pouze mezi sousedními snímky. To má za následek velkou nekonzistenci při generování dlouhých videí. Z těchto důvodů jsem se rozhodl tuto techniku nepoužít.

AnimateDiff[6] Metoda AnimateDiff spočívá ve vložení pohybového adaptéru do zmrazeného základního modelu generující obrázky z textu. Takto je celá síť natrénována na videích a výsledkem je pohybový adaptér, který je možné vložit do existujících odvozených modelů. Díky tomu všechny odvozené modely vycházející ze stejného základního modelu mohou generovat rozmanité animace. Tento popis vychází z abstraktu AnimateDiff[6].

AnimateDiff je technika použitá v této práci a proto je detailně rozebrána níže. Hlavní nevýhodou je schopnost generovat pouze 16 snímků videa, ale tento problém je vyřešen pomocí jiné techniky, která je také popsána níže.

2.2.3 Porovnání metod generování videa

Porovnání vyše rozebraných metod se nachází v tabulce 2.1.

Tabulka 2.1: Porovnání metod generování videa

Název metody	Max. délka	Rozlišení videa	Personalizace	ControlNet
ModelScopeT2V	∞[2]	1024×576 px	✗	✗
Stable Video Diffusion	24 snímků	1024×576 px	✓	✗
Show-1	29 snímků	576×320 px	✗	✗
I2VGenXL	16 snímků	1280×720 px	✓	✗
Text2Video-Zero	∞[2]	512×512 px[1]	✓	✓
AnimateDiff	16 snímků	512×512 px[1]	✓	✓

[1] Rozlišení je možné změnit, 512×512 pixelů je pouze standardní rozlišení Stable Diffusion 1.5 modelu.

[2] Praktická maximální délka je samozřejmě omezená pamětí.

2.3 Personalizace modelů

Základní modely (anglicky *foundational models*) dokáží generovat různorodá videa a obrázky, ale často chceme model nějakým způsobem personalisovat. Personalisace modelů znamená, že je model schopný generovat například nějaký konkrétní styl nebo postavu. Díky tomu je možné dosáhnout lepší vzhledové konzistence při generování subjektu v různých situacích, což je při generování videa klíčové. Existuje několik metod, jakými můžeme model personalizovat, každá je vhodná na jiné použití. Jejich přehled je uveden níže.

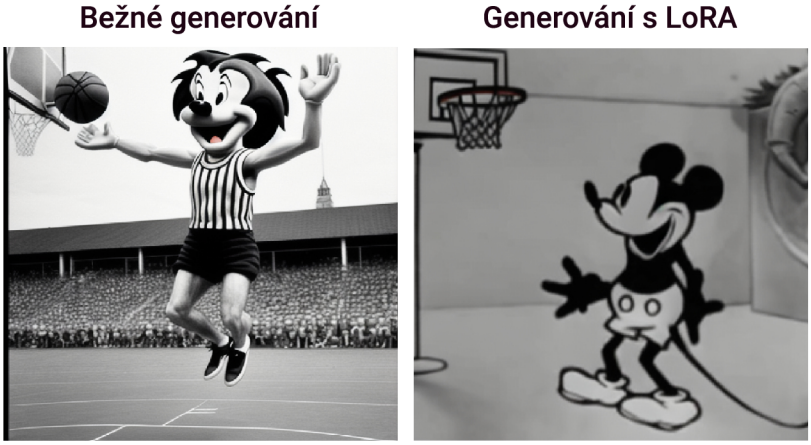
Dolaďování modelu Metoda dolaďování modelu (anglicky *model fine-tuning*) spočívá v dotrénování základního modelu vlastními daty. Tato metoda je velice nákladná a vyžaduje spoustu dat. V práci jsou použité dotrénované modely na bázi Stable Diffusion 1.5.

LoRA[7] LoRA (*Low-Rank Adaptation of Large Language Models*) je způsob, jakým je možné upravit chování již natrénovaného modelu bez zásahu do jeho původních parametrů. Funguje tak, že se do modelu vloží menší počet nových parametrů a pouze ty se natrénují pro specifické potřeby.

Jak název napovídá, původně byla tato metoda navržena pro doladění velkých jazykových modelů, avšak je možné jí použít i pro difúzní modely. Tuto funkcionalitu poskytuje například knihovna PEFT[13], která je použita ve finální implementaci. Porovnání vygenerovaných obrázků s použitím a bez použití LoRA je na obrázku 2.6.

IP adaptér[25] IP adaptér umožňuje ovlivňovat výstup latentního difúzního modelu pomocí jednoho, nebo několika obrázků. Existuje více variant adaptéru, jeden řídí například celkový styl a další pouze modifikuje obličej.

Textová inverze Textová inverze (anglicky *textual inversion*) je metoda pro personalizaci modelů, která funguje na principu učení se nových slov na základě několika příkladových obrázků. Výsledný soubor je narozdíl od ostatních technik velmi malý (několik KB). Naučená reprezentace je vložena do textového enkodéru a lze ji poté použít.



Obrázek 2.6: Výstup difúzního modelu Stable Diffusion 1.5 v kombinaci s LoRA modelem. Prompt „animated man in suit standing on a beach, best quality, masterpiece“

Porovnání metod Porovnání metod personalizace modelů je uvedeno v tabulce 2.2:

Tabulka 2.2: Porovnání metod personalizace modelů

Název metody	**Velikost souboru**	**Trénování**[1]	**Úroveň personalizace**
Doladování modelů	~ 5 GB	✓	Velká
LoRA	~ 100 MB	✓	Velká
IP adaptér	~ 10 MB[2]	✗	Střední
Textová inverze	~ 5 kB	✓	Malá

[1] Informace, zda je třeba provádět trénování pro vytvoření nového stylu. Trénování pomocného modelu (například IP adaptéru) se v tomto kontextu nebere do úvahy, protože je třeba natrénovat pouze jednou.

[2] Uvedená velikost je přibližná velikost příkladových obrázků. Použití IP adaptéru vyžaduje navíc ještě předtrénovaný model o velikosti 50 MB – 1 GB.

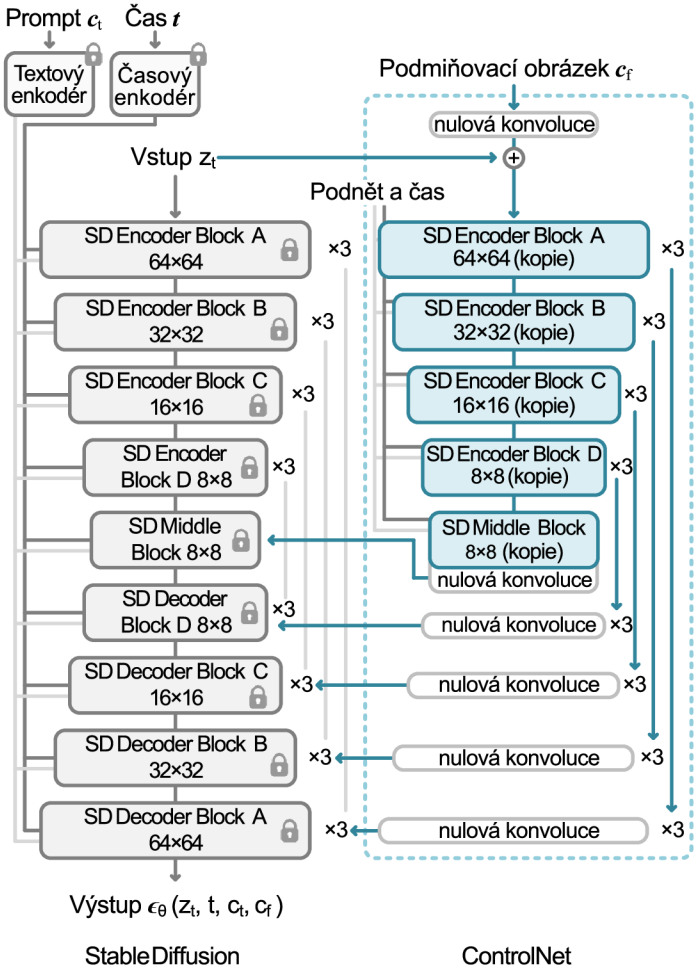
2.3.1 Kontrola nad generováním

ControlNet Architektura ControlNet[28] umožnuje pomocí dalšího obrázku kontrolovat generovaný obsah. Schéma fungování je na obrázku 2.7. Příklad s použitím a bez použití sítě ControlNet je na obrázku 2.8.

T2I adaptér[14] T2I adaptér umožňuje to stejné jako síť ControlNet. Na základě obrázku dokáže kontrolovat, jak bude obsah vygenerován. Hlavní výhodou je, že se adaptér vyhodnocuje pouze jednou, narozdíl od ControlNetu, kde se síť vyhodnocuje každý odšumovací krok. To zajišťuje větší rychlost na úkor přesnosti.

2.4 Pohybový adaptér AnimateDiff

Pohybový adaptér AnimateDiff[6] funguje na principu vložení nové naučené sítě do existující odšumovací sítě U-Net. Princip fungování AnimateDiff je naznačen na obrázku 2.9.



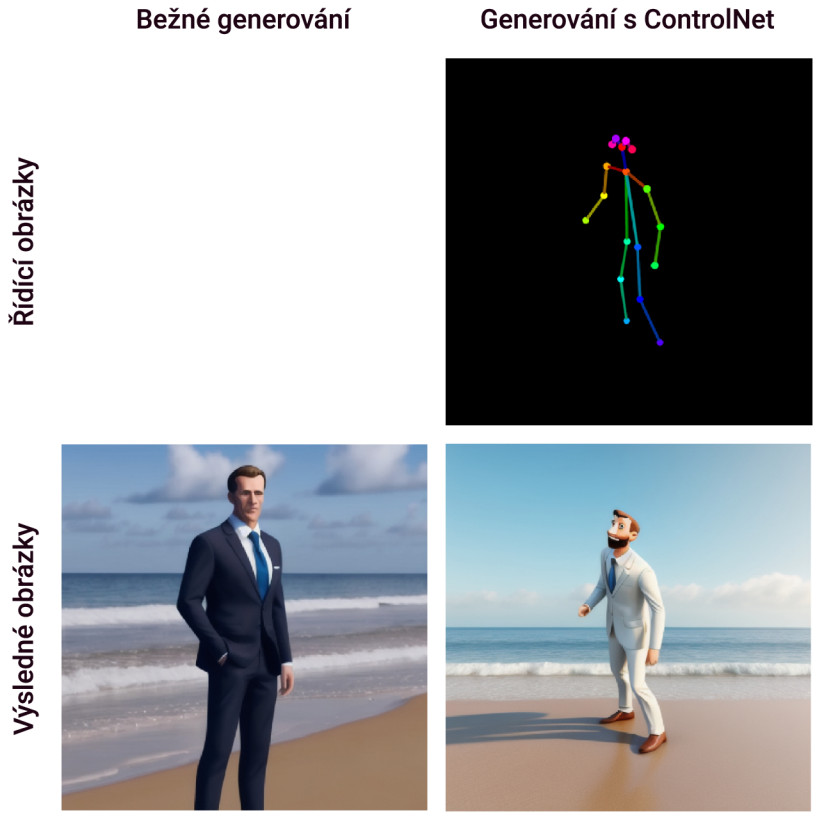
Obrázek 2.7: Schéma architektury ControlNet. Převzato z [28] (Upraveno).

2.4.1 Zrychlení generování

Původní AnimateDiff adaptér vyžaduje asi 20 kroků odšumnění k dosažení přijatelných výsledků. Při generování krátkých videí to není nijak zatěžující, ale pro generování delších animací jako v této práci už je to znát. Momentálně existují dvě metody pro zmenšení počtu potřebných kroků:

LCM LoRA[12] LCM (*Latent consistency model*) LoRA je adaptér pro Stable Diffusion modely. Funguje na tom principu, že ze zašumněného obrázku (případně videa) se snaží ihned vytvořit čistou variantu. Následkem tohoto principu difúzní model produkuje přijatelné obrázky už po několika málo krocích za cenu snížení kvality.

AnimateDiff lightning[10] AnimateDiff lightning je narozdíl od LCM LoRA vlastní model, který nahrazuje model AnimateDiff. Výsledkem je ale to stejné, difúzní model dokáže produkovat čisté animace už po několika málo krocích. AnimateDiff lightning existuje ve třech variantách: `2-step`, `4-step` a `8-step`. Z jejich názvu vyplývá, po jakém počtu kroků



Obrázek 2.8: Výstup difúzního modelu Stable Diffusion 1.5 v kombinaci se sítí ControlNet[28]. Prompt „animated man in suit standing on a beach, best quality, masterpiece“

konvergují. Po otestování obou variant jsem se rozhodl v této práci využít tuto metodu z důvodu vyšší kvality vygenerovaných animací.

2.4.2 Motion LoRA

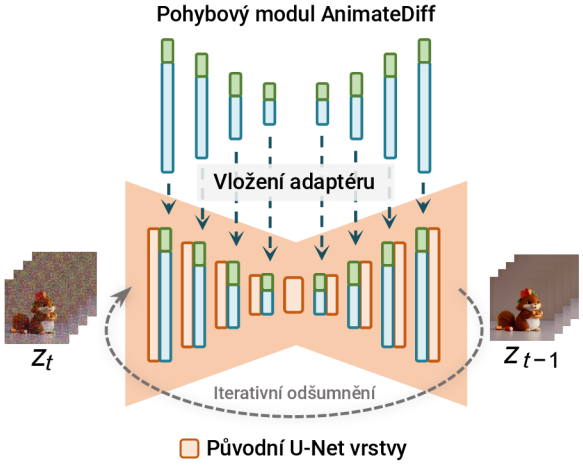
Motion LoRA je LoRA adaptér pro pohybový adaptér AnimateDiff. Umožňuje kontrolovat pohyb virtuální kamery. Existuje několik předtrénovaných modelů přímo od autorů AnimateDiff. Bohužel autoři pravděpodobně tyto adaptéry natrénovali na videích s vodoznakem, takže při použití adaptéru může být viditelný vodoznak ve vygenerovaném videu. Příklad Motion LoRA je ukázán na obrázku 2.10.

2.4.3 Generování dlouhých videí

Pohybový adaptér AnimateDiff[6] je navržen pouze pro generování 16 snímků. Díky technice FreeNoise[16] lze generovat videa delší při zachování temporální konzistence. Tato technika se skládá ze dvou částí:

- úpravou počátečního latentního šumu
- použití posuvného okna pro pohybový modul

Technika FreeNoise je implementována pro několik různých modelů. Jedním z nich je právě pohybový modul AnimateDiff. Vychází z [16].



Obrázek 2.9: Princip pohybového adaptéru AnimateDiff

Obrázek 2.10: Ukázka AnimateDiff Motion LoRA `pan-left`. Snímky byly převzaty z [6].

2.5 Procedurální animace

Procedurální animace je typ počítačové animace pro generování animace. Na rozdíl od klasických animací, které jsou vytvořeny předem a zahrnují specifické pohyby pro každou akci, procedurální animace umožňují generování pohybů postav dynamicky.

Základním principem procedurální animace je využití simulace fyzikálních zákonitostí pro generování realistických pohybů postav. Jednou z nejčastějších technik je využití fyzikálně založených animací, které se opírají o simulaci tuhých těles. Tento typ simulace, běžně používaný v herních enginech jako Unity nebo Unreal, umožňuje postavám reagovat na okolní prostředí a na interakce v reálném čase.

Jedním z typických příkladů je animace vody, která místo ručního animování může být generována na základě simulace fluidní dynamiky. Tento přístup jednak přináší vyšší stupeň realismu, ale také efektivněji řešit různorodé a nečekané situace, které mohou nastat.

Existuje několik způsobů generování lidského pohybu pomocí procedurální animace:

Ragdoll fyzika Ragdoll fyzika funguje na principu, že vytváří humanoidní tělo s klouby, které napodobují skutečné stupně svobody. Umožňuje simulaci pádu člověka pomocí simulace tuhých těles. Toto řešení umožňuje generovat realistické pády a interakce postav s prostředím, což by bylo obtížné dosáhnout pomocí předdefinovaných animací. Hlavní nevýhodou je nepředvídatelnost, která může vést k nechtěným komickým situacím.

Simulace tuhých těles Hlavní problém s ragdoll systémy je absence motorické kontroly, což způsobuje, že postavy pouze padají, místo aby chodily nebo skákaly. Tento problém lze překonat vynucením pozic některých částí (například rukou a nohou) a zbytek řídit ragdoll omezeními.

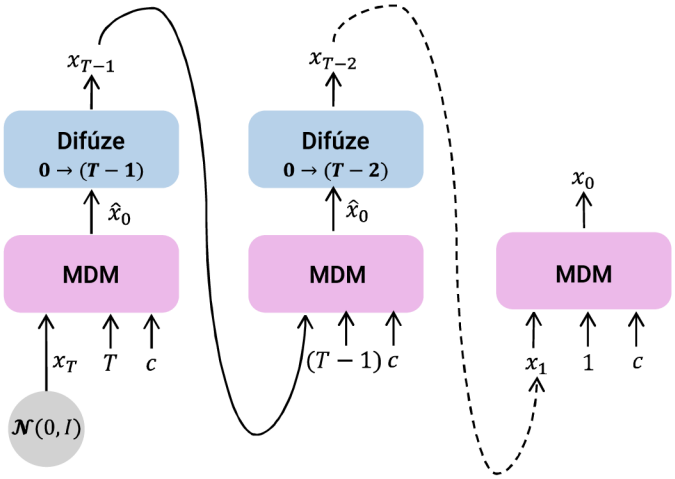
Inverzní kinematika V inverzní kinematice pouze specifikujeme cílové body a pohyb se dopočítá automaticky. Je třeba mít specifikované stupně svobody u kloubů, stejně jako u ragdollu. Tato technika produkuje realistické pohyby.

Sekce byla převzata z [30]. Tato práce nevyužívá pro generování lidského pohybu klasickou metodu procedurální animace, ale namísto toho používá speciální pohybový difúzní model rozebraný níže.

2.5.1 Pohybový difúzní model

Pohybový difúzní model (*MDM – Motion diffusion model*) umožňuje generovat přirozený lidský pohyb o délce několika sekund. Model generuje data v upraveném formátu SMPL[11] (*A Skinned Multi-Person Linear Model*). Původní SMPL formát obsahuje 24 bodů, kdežto model jich produkuje pouze 22. Z nedohledatelného důvodu chybí pozice dlaní, to však není pro tuto práci podstatné.

Fungování pohybového difúzního modelu je znázorněno na obrázku 2.11. Nejprve je získáno podmínění c z textového enkodéru na základě textového popisu. Poté je vygenerován náhodný šum x_T o rozměrech požadovaného pohybu. Následně probíhá iterace od T do 1, kde T je počet kroků difúze. V každém kroku t model ze zašumněného vzorku předpovídá čistý vzorek $\hat{x}_0$ a následně probíhá difúze zpět na x_{t-1}. Odšumovací model je založen na bázi architektury transformer, narozdíl od latentního difúzního modelu popsaného výše, který na odšumování používá architekturu U-Net. Vysvětlení vychází z [21].



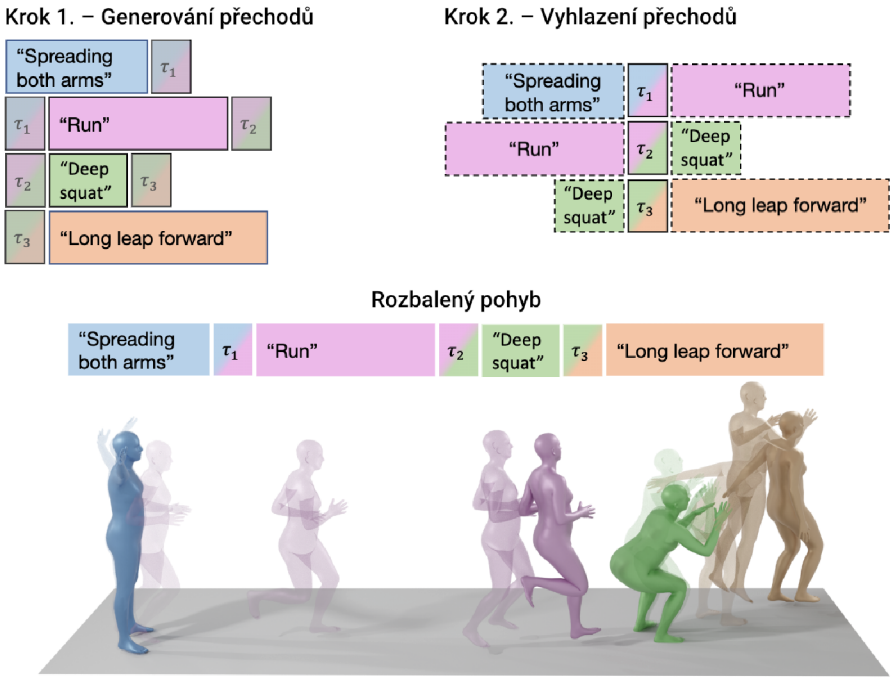
Obrázek 2.11: Fungování pohybového difuzního modelu. Převzato z [21] (Upraveno).

2.5.2 Generování dlouhých pohybů

Pohybový difúzní model stejně jako AnimateDiff produkuje omezenou délku pohybů. Navíc by bylo dobré, kdyby bylo možné mít několik akcí zřetězených za sebou. Přesně tento problém řeší technika priorMDM[20]. Technika priorMDM se zaměřuje na několik věcí:

- Generování dvou osob současně (*ComMDM*)
- Generování pohybu podle dané trajektorie
- Generování dlouhých pohybů (*DoubleTake*)

Technika DoubleTake, jak název napovídá, se skládá ze dvou kroků. V Prvním kroku se generují samostatné pohyby s pevným prefixem získaným z předchozího pohybu (kromě prvního pohybu). Tímto způsobem je už vygenerován dlouhý pohyb o několika akcích, avšak přechody mohou být nepřirozené. Z toho důvodu následuje druhý krok, ve kterém v místech přechodů probíhá částečná difúze. Popis techniky je převzatý z [20]. Fungování je znázorněno na obrázku 2.12.



Obrázek 2.12: Znázornění metody priorMDM. Převzato z [20] (Upraveno).

2.5.3 OpenPose

OpenPose[3] je systém pro rekonstrukci pozice lidí z 2D obrázků. V této práci je tomu přesně naopak. Z OpenPose reprezentace se pomocí difúzního modelu generují snímky videa. Toho je dosáhnuto pomocí sítě ControlNet, která byla natrénována na obrázcích lidí a příslušnou odhadnutou OpenPose kostrou.

Kapitola 3

Zhodnocení současného stavu a požadavky na řešení

Generování videa z textového popisu je momentálně velmi populární téma. Každý měsíc vychází nová metoda, nebo vylepšení existující metody pro generování videa. Tato práce se nesnaží přinést novou metodu pro generování videa, ale propojit metody existující k vytvoření delších animovaných příběhů.

3.1 Existující služby a řešení

V této sekci jsou popsány existující služby pro automatické generování videí a podobného obsahu z textového popisu. Krátké generátory několika-vteřinových videí jako je například Gen-2 od Runway nejsou zahrnuty.

- https://pictory.ai/ Umožňuje generování komentovaných videí podle scénáře. Video je komentované pomocí umělé inteligence převádějící text na řeč doplněné o hudbu z hudební banky. Jednotlivé části videa jsou pouze automaticky získávány z videobanky, nejsou generované žádným modelem.
- https://invideo.io/ Funguje prakticky stejně jako https://pictory.ai/. Přidanou hodnotou je možnost zahrnout vygenerované obrázky pomocí AI. Zároveň umožňuje scénář generovat pomocí jazykového modelu.
- https://www.synthesia.io/ Slouží pro generování virtuální mluvících avatarů. Vstupem je text, výstupem je video na kterém je vygenerovaná osoba, která tento text čte. Na výběr je omezený počet avatarů (asi 160), ale je možné nechat si vytvořit digitální kopii sebe.
- https://www.magicslides.app/ Umožňuje z textového popisu a dalších podkladů vytvořit prezentaci Microsoft Powerpoint/Google slides.
- https://openai.com/sora OpenAI Sora je momentálně nejpokročilejší nástroj pro generování videa difúzním modelem. Umožňuje generovat až minutu dlouhá videa. Zatím není nástroj otevřen veřejnosti, takže se jeho skutečné kvality nedají dost dobře posoudit.

3.2 Požadavky na řešení

Program by měl umožnit uživateli jednoduše vygenerovat animaci z jeho textového popisu. To obnáší několik požadavků:

Uživatelské rozhraní Uživatelské rozhraní bude implementováno formou webového rozhraní. Uživatel by měl být schopný tvořit videa bez znalosti detailů fungování systému. Rozhraní bude primárně sloužit pro sbírání dat k uživatelské studii. Cílem rozhraní bude jednoduchost používání. Bude obsahovat vstupní textové pole pro popis videa, vlastní komponentu pro editaci detailního scénáře a výstupní video.

Generování scénáře Z krátkého textového popisu bude možné pomocí velkého jazykového modelu vygenerovat podrobný scénář popisující vzhled prostředí, vzhled osob a jednotlivé akce. Tento scénář bude moci uživatel následně editovat v uživatelsky přívětivém prostředí.

Nenáročnost na prostředky Program by měl fungovat na jedné grafické kartě s 24 GB video paměti. Několikaminutové video by se nemělo generovat déle než jednu hodinu.

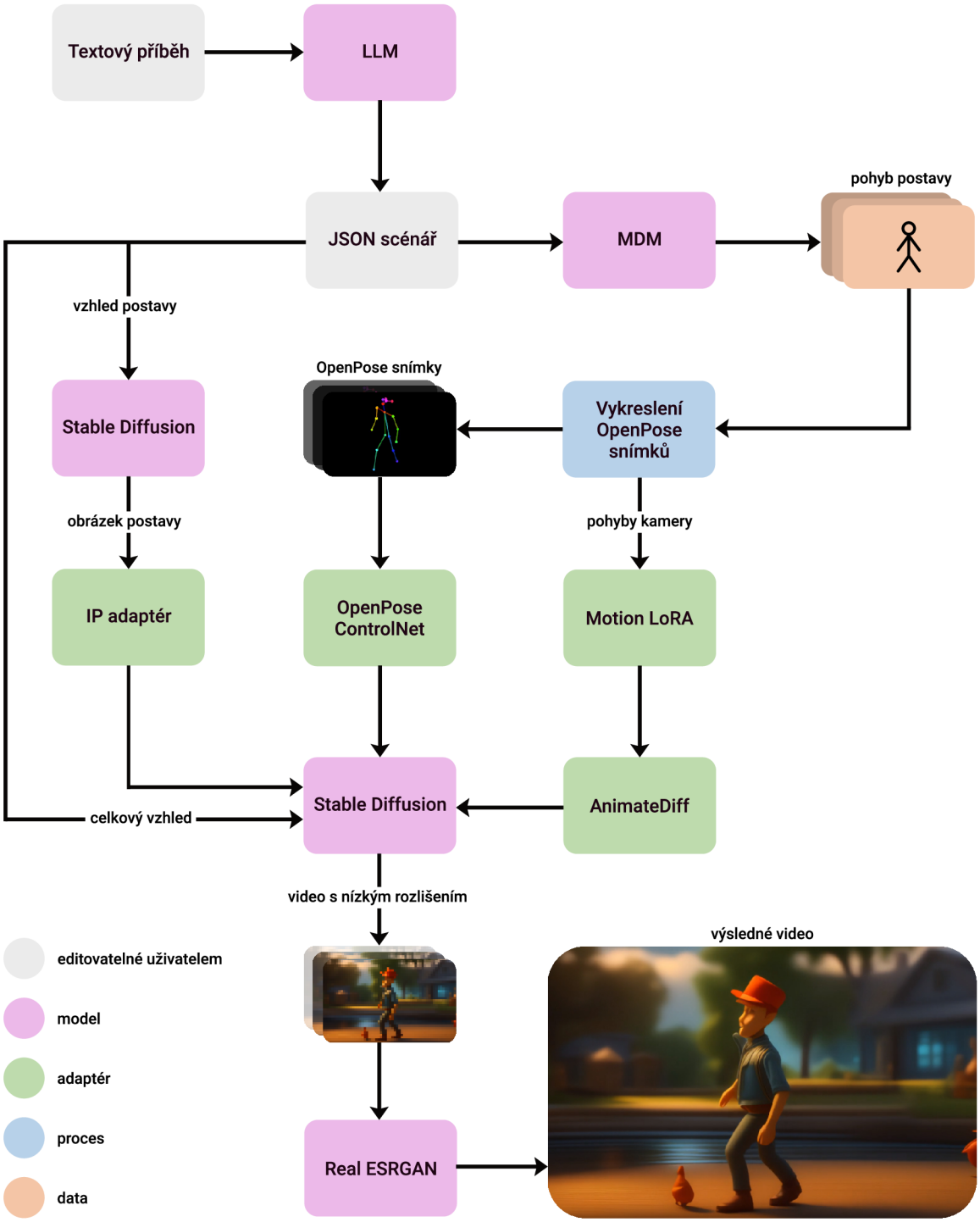
Konzistentní postavy, prostředí a celkový vzhled Výsledná animace by měla mít dobrou temporální konzistenci. Mezi jednotlivými snímky by nemělo docházet k příliš razantním změnám, které by mohly vést například k problikávání obrazu. Vzhled postav a prostředí by se měl měnit jenom minimálně, ideálně vůbec. Po dobu celého videa by měl být udržený podobný vizuální dojem. To znamená, že například z 2D animace by se neměla stát 3D animace.

Formát videa Výstupní video bude ve formátu MP4 s poměrem stran 16:9 a rozlišením Full HD (1920×1080 pixelů). Snímková frekvence bude 10 snímků za sekundu. Délka videa bude moci dosahovat i několika minut.

3.3 Omezení řešení

Řešení mé práce se omezí na příběhy o jedné postavě v záběru. Různé postavy mohou být v různých scénách, ale v jedné scéně bude moci být maximálně jedna postava. Více postav v jedné scéně by mohlo být řešeno v navazující práci. Bylo by třeba řešit interakci postav při generování pohybu, udržení obou postavu v jednom záberu a podobně. Proto jsem se rozhodl zaměřit pouze na videa s tímto omezením.

Dále jsem se rozhodl vynechat titulky a generování mluveného slova. Přinášelo by to další zbytečnou komplexitu.



Obrázek 3.1: Schéma implementace

Kapitola 4

Implementace

V této kapitole je popsaná celá implementace nástroje s důrazem na zajímavé části a řešení vzniklých problémů. Implementace je provedena primárně v jazyce Python s použitím knihoven přestavených v následujících sekcích. Část uživatelského rozhraní je implementována ve webových technologiích. Implementace se skládá z několika relativně samostatných částí, které jsou postupně popsány níže. Propojení jednotlivých částí implementace je znázorněno na obrázku 3.1.

4.1 Scénář videa

Scénář videa je koncipován jako soubor ve formátu JSON. Formát scénáře je definovaný podle specifikace JSON Schema[24]. Tento formát umožňuje dvě věci. První, pravděpodobně zřejmé využití je validace scénáře. Je to mnohem robustnější řešení než validovat JSON soubor ručně. Druhé využití má schéma při generování scénáře, které je popsáno níže. Příklad kompletního scénáře je na paměťovém médiu. Scénář je podle uvedeného schématu (4.1) ověřován Python knihovnou `jsonschema`. Knihovna zpřístupňuje funkci `validate`, která přijímá dva argumenty. Prvním argumentem je instance ověřovaného objektu a druhým argumentem je ověřovací JSON schéma. Pokud je JSON nevalidní, funkce vyvolá výjimku `ValidationError`. V opačném případě je struktura objektu ve formátu JSON validní.

4.1.1 Struktura scénáře

Scénář se skládá ze scén. Scéna obsahuje popis postavy a seznam akcí. Popis postavy je volitelný, pokud není uveden, tak je scéna bez postavy. Popis postavy může být například: „Guy in his mid-30s, with short, curly hair, wearing a bright red shirt, denim jeans and comfortable walking shoes“. Každá scéna má minimálně jednu akci. Jednotlivé akce obsahují tři atributy: délku akce, popis pohybu a popis vzhledu scény. Délka akce je vyjádřena jako celé číslo a jeho jednotka jsou sekundy. Popis pohybu vyjadřuje pohyb postavy, například: „The person is walking slowly, occasionally stopping“. Popis scény je popis vzhledu scény bez postavy, například: „Park with the colors of various flowers and the lush green of the grass. The sun is shining brightly, casting playful shadows through the leaves of tall trees“.

Lze si povšimnout, že všechny popisy jsou v anglickém jazyce. To je z toho důvodu, že textové enkodéry jak pohybového modelu, tak latentního difúzního modelu jsou naučeny na anglických textech. Při generování scénáře lze však použít i jiný jazyk (například češtinu) — velký jazykový model vygeneruje příslušný scénář v angličtině.

Obrázek 4.1: Schéma scénáře

```
{
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "character_description": {"type": "string"},
            "actions": {
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "length": {"type": "number"},
                        "motion_description": {"type": "string"},
                        "scene_description": {"type": "string"}
                    },
                    "required": [
                        "length",
                        "motion_description"
                    ]
                }
            }
        },
        "required": ["actions"]
    }
}
```

4.1.2 Generování scénáře

Scénář lze vygenerovat z textového popisu pomocí velkého jazykového modelu. Generování scénářů je zajištěno knihovnou `llama-cpp-python`[1], což je wrapper nad programem `llama.cpp`[2]. Tato knihovna umožňuje omezit generování textu pomocí bezkontextových gramatik. To se výborně hodí pro generování validních dokumentů ve formátu JSON. Bezkontextová gramatika je automaticky vygenerována z JSON Schema uvedeného výše, tuto funkcionalitu rovněž zajišťuje knihovna `llama-cpp-python`. Je však třeba zajistit, aby bylo generování dokumentu dokončeno (pokud se při generování narazí na maximální počet tokenů), protože to není nijak implicitně zajištěno. K generování scénáře je použitý model `QuantFactory/Meta-Llama-3-8B-GGUF`[3], konkrétně varianta `Meta-Llama-3-8B.Q6_K.gguf`.

Lepších výsledků bylo dosaženo uvedením příkladu transformace textového příběhu na scénář. Příklad se nachází před vlastním promptem. Tato technika se nazývá *one-shot prompting*. Zároveň je uvedeno vysvětlení jednotlivých JSON atributů pro lepší pochopení zadání modelem.

[1] https://github.com/abetlen/llama-cpp-python
[2] https://github.com/ggerganov/llama.cpp
[3] https://huggingface.co/QuantFactory/Meta-Llama-3-8B-GGUF

Výsledný vygenerovaný scénář je validován postupem uvedeným výše. Délky akcí jsou omezeny do validních hodnot.

4.2 Generování pohybu člověka

Generování pohybu humanoidní kostry je implementováno pomocí upravené oficiální implementace techniky priorMDM[20][4]. Tato práce konkrétně využívá techniku *DoubleTake* (vysvětlenou v sekci 2.5.2), která umožňuje generovat několik akcí plynule navazujících na sebe.

4.2.1 Úpravy implementace priorMDM

Oficiální implementace slouží pro replikování publikovaných výsledků, ne jako knihovna použitelná pro další projekt. Byly dvě možnosti jak k tomuto problému přistoupit. Prvním řešením by bylo implementovat tento model do knihovny Diffusers. Zvolil jsem druhou cestu, kterou je úprava existující implementace. Hlavní potřebnou změnou bylo změnit absolutní cesty importů na relativní cesty. Toho bylo částečně automaticky dosaženo pomocí programu `abs2rel`[5]. Tento program však nepokrývá všechny případy, proto bylo třeba některé cesty ručně opravit. Konkrétní příklad úpravy:

```
from model.comMDM import ComMDM -> from ..model.comMDM import ComMDM
```

Další změnu vyžadovalo načítání modelů. V původní implementaci se program spoléhá na to, že je spuštěn přímo, ne z podadresáře — cesty jsou definovány vzhledem ke kořenu projektu. Problém jsem vyřešil za pomocí funkcí `join` a `dirname`, které se nachází ve standardní knihovně jazyka Python, konkrétně v modulu `os.path`. Vestavěný atribut modulu `__file__` obsahuje celou cestu k danému modulu (souboru). Po zavolání funkce `dirname(__file__)` se vrátí cesta ke složce, kde se nachází daný soubor. Funkce `join` umožňuje korektně sloučit několik cest do jedné. Konkrétní příklad převodu je následující:

```
args.model_path -> join(dirname(__file__), '..', args.model_path)
```

Velkou změnou bylo odstranění nutnosti načítání datasetu. Implementace priorMDM vyžaduje HumanML3D[6] dataset. Tento dataset je nutný pro trénování a evaluaci. V základní implementaci je třeba i na běžné spouštění modelu, což ale nedává úplně smysl. Je zbytečné načítat obrovský dataset (přes 5 GB), který se poté nepoužije, navíc dataset je licencovaný tak, že ho není možné volně distribuovat — je potřeba ho pomocí pokynů v repozitáři zrekonstruovat. Z těchto důvodů jsem se rozhodl nutnost datasetu odstranit. Při analýze původního kódu jsem zjistil, že ve skutečnosti pro generování jsou potřeba pouze soubory `Mean.npy` a `Std.npy`, které slouží pro rekonstrukci pohybu z reprezentace vygenerované modelem. Kód pro rekonstrukci byl vyextrahován do samostatné funkce a původní funkce pro načítání datasetu se již nevolá, tím byl tento problém vyřešen.

Posledním velkým problémem byl fakt, že původní program používal starší závislosti oproti zbytku mé implementace. Naštěstí byla problémová pouze jedna závislost, které počítala se starou verzí numpy. K dosažení zpětné kompatibility bylo zajištěno pomocí vytvoření vlastních importů. Tento postup je ukázán na obrázku 4.2.

V neposlední řadě byly nahrazeny volání funkce `print` řádným logováním pomocí modulu `logging` ze standardní knihovny jazyka Python.

[4] https://github.com/priorMDM/priorMDM
[5] https://github.com/KennedyRichard/abs2rel
[6] https://github.com/EricGuo5513/HumanML3D

Obrázek 4.2: Dosažení zpětné kompatibility u knihovny Numpy

```
import sys

sys.modules["numpy.bool"] = bool
sys.modules["numpy.int"] = int
sys.modules["numpy.float"] = float
sys.modules["numpy.complex"] = complex
sys.modules["numpy.object"] = object
sys.modules["numpy.str"] = str
sys.modules["numpy.unicode"] = str
```

4.2.2 Vlastní syntézní script

Původní syntézní script se nachází v `sample/double_take.py`. Je navržen tak, aby se dal přímo volat z příkazové řádky. Můj syntézní script přebírá velkou část kódu z původního programu, ale jsou odstraněny některé nepotřebné části.

Implementace generování pohybu je ve funkci s názvem `generate_motion`. Tato funkce má jediný argument `texts` — seznam dvojic s textovým popisem pohybů a délkou příslušných pohybů. Funkce vrací vygenerované pohyby a skutečné délky vygenerovaných pohybů. Implementace generování pohybu se nachází v `priorMDM/infer.py`.

Technika DoubleTake počítá alespoň s dvěma různými pohyby. V případech, kdy je ve scéně pouze jedna akce, je tato akce zdvojena a po vygenerování je použita první polovina.

4.3 Vykreslení řídících snímků

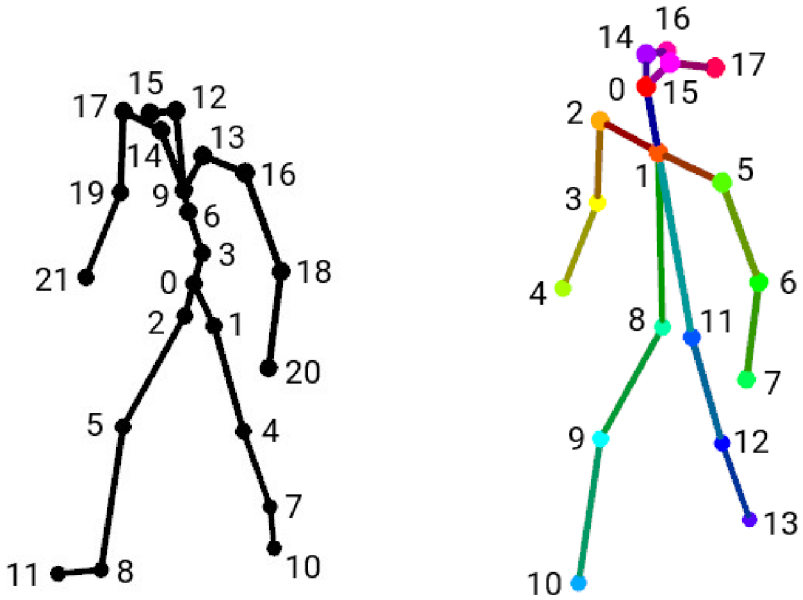
Typicky jsou řídící snímky OpenPose získávány detekcí z již existujícího videa. V této práci se pohyb generuje pomocí difúzního modelu, proto je třeba nějak tyto snímky vygenerovat. To obnáší několik překážek, jejichž řešení je popsáno v následujících sekcích.

4.3.1 Převod reprezentace SMPL na OpenPose

MDM produkuje data ve formátu SMPL. Z tohoto důvodu je potřeba před samotným vykreslením převést formát data do formátu OpenPose. Tento převod je inspirován článkem [5]. Vizuální porovnání obou reprezentací je znázorněno na obrázku 4.3. Jak je z obrázku patrné, reprezentace OpenPose má navíc klouby a spojení (kosti), které je třeba dopočítat.

Pro přehlednost budou body reprezentace SMPL značeny S_i a body reprezentace OpenPose O_i, kde i je index kloubu příslušné reprezentace. Většina kloubů jde přímo namapovat z jedné reprezentace do druhé viz tabulka 4.1. Zbylých 5 kloubů je třeba dopočítat pomocí lineární algebry. Nejprve je dopočítána pozice kloubu O_1 (spodní část krku):

$$O_1 = \frac{S_{13} + S_{14}}{2} \tag{4.1}$$



Obrázek 4.3: Porovnání reprezentací SMPL (vlevo) a OpenPose (vpravo)

Tabulka 4.1: Mapování SMPL na OpenPose

OpenPose index	0	1	2	3	4	5	6	7	8
SMPL index	12	-	17	19	21	16	18	20	2
OpenPose index	9	10	11	12	13	14	15	16	17
SMPL index	5	8	1	4	7	-	-	-	-

Poté je spočítán normalizovaný normálový vektor plochy tvořený bodem O_1, S_{12} a S_{15}. Tento vektor udává směr od středu hlavy k uchu. Nechť je tento vektor značen $\vec{v_N}$:

$$\begin{aligned} \vec{v_1} &= S_{12} - O_1 \\ \vec{v_2} &= S_{15} - O_1 \\ \vec{v_N} &= \frac{\vec{v_1} \cdot \vec{v_2}}{|\vec{v_1} \cdot \vec{v_2}|} \end{aligned} \tag{4.2}$$

Poté je třeba spočítat normalizované vektory: od krku k vršku hlavy ($\vec{v_U}$), od středu hlavy do zadní části hlavy ($\vec{v_B}$):

$$\begin{aligned} \vec{v_U} &= \frac{\vec{v_1}}{|\vec{v_1}|} \\ \vec{v_B} &= -\frac{\vec{v_N} \cdot \vec{v_U}}{|\vec{v_N} \cdot \vec{v_U}|} \end{aligned} \tag{4.3}$$

Nakonec je je možné dopočítat pozice očí a uší — O_{14}, O_{15}, O_{16} a O_{17}:

$$\begin{aligned} O_{15} &= S_{12} + \frac{\vec{v_N}}{12} + \frac{3}{20}\vec{v_B} + \frac{\vec{v_U}}{20} \\ O_{16} &= S_{12} - \frac{\vec{v_N}}{12} + \frac{\vec{v_B}}{20} \\ O_{17} &= S_{12} + \frac{\vec{v_N}}{12} + \frac{\vec{v_B}}{20} \\ O_{18} &= S_{12} - \frac{\vec{v_N}}{12} + \frac{3}{20}\vec{v_B} + \frac{\vec{v_U}}{20} \end{aligned} \tag{4.4}$$

Konstanty byly získány empiricky. Tento převod je implementován v souboru `conditioning.py` s použitím matematické knihovny Numpy.

4.3.2 Použitá knihovna na vykreslování

Vykreslení jednotlivých OpenPose snímků reprezentujících pozici postavy je realizováno pomocí knihovny Mayavi[18]. Mayavi je knihovna pro vizualizaci vědeckých dat a 3D kreslení v programovacím jazyce Python. Umožňuje vykreslování jak pokročilých 3D grafů, tak primitivních objektů.

Knihovna Mayavi byla použita z důvodu skutečného 3D vykreslování narozdíl od knihovny Matplotlib. Matplotlib nerespektuje pořadí objektů v 3D prostoru, to má za následek nesprávné vykreslení v grafu[7].

Běžně se používá ve formě GUI, jako interaktivní prostředí pro vykreslování, ale lze přepnout do režimu kdy všechno vykreslování probíhá mimo obrazovku (anglicky *offscreen rendering*). Toho lze dosáhnout nastavením `mlab.options.offscreen` na hodnotu `True`. To však není k provozu na serveru bez obrazovky dostatečné. Mayavi pro vykreslování využívá knihovnu VTK (zkratka *The Visualization Toolkit*). Samotné VTK využívá pro vykreslování OpenGL. OpenGL kontext nelze otevřít bez běžícího X-serveru. Řešením je použít balíček `Xvfb` (*virtual framebuffer X server*), který vytvoří virtuální X server. Tuto funkcionalitu v Pythonu poskytuje knihovna `pyvirtualdisplay`[8]. Virtuální obrazovka je vytvořena například pomocí volání `display = Display(visible=0, size=(1280, 1024))` a spuštěna pomocí `display.start()`. Jelikož program vykresluje mimo obrazovku, tak rozlišení virtuální obrazovky není důležité.

Tělo virtuálního humanoida se skládá z kloubů a kostí. Klouby jsou reprezentovány jako koule (v Mayavi tomu odpovídá funkce `points3d`). Kosti jsou reprezentovány pomocí válce (v Mayavi tomu odpovídá funkce `plot3d`). Z důvodu rychlosti je nejprve vytvořena kostra a poté jsou pouze manipulovány pozice jednotlivých kostí a kloubů. Mayavi ve výchozím nastavení překresluje scénu při každé změně. To je však v tomto případě nežádoucí, proto bylo třeba toto chování upravit. Změnou proměnné `figure.scene.disable_render` lze vypínat a zapínat vykreslování — při změně pozice kloubů a kostí je vykreslování vypnuto a poté je znovu zapnuto.

Každý snímek je zachycen příkazem `mlab.screenshot()`, který vrací aktuální vykreslený stav scény jako pole bajtů. S obrázkem jako polem bajtů by se špatně pracovalo, proto je tato reprezentace převedena funkcí `Image.fromarray` z knihovny PIL do vhodnější reprezentace.

[7] O tomto problému se lze dočíst v Matplotlib FAQ — https://matplotlib.org/stable/api/toolkits/mplot3d/faq.html

[8] Další řešení jsou uvedeny v dokumentaci knihovny Mayavi — Tips and Tricks — https://docs.enthought.com/mayavi/mayavi/tips.html

4.3.3 Pohyb virtuální kamery

Kostra humanoida se přirozeně bude pohybovat v prostoru. Proto je třeba vyřešit, jak docílit toho, aby subjekt nevyšel z virtuálního záběru. Přímočará možnost je nechat kameru neustále kopírovat pohyb kostry, čímž by postava byla neustále uprostřed snímku (či jiném fixním bodu). Toto řešení se možná na první pohled nezdá jako špatné, ale má určité vady. První vadou je nízká dynamičnost vygenerovaných záběrů, subjekt ve filmu také není stále uprostřed obrazovky. Druhý problém je způsoben technikou generování videa. Pohybový adaptér AnimateDiff špatně generuje chůzi (případně běh a podobně), když se subjekt nehýbe v rámci 2D prostoru. Místo chození v prostoru to vypadá jako chůze na místě.

Z těchto důvodů byl zvolen jiný přístup. Subjektu je dána volnost pohybovat se volně mimo střed a virtuální kamera se začně hýbat až poté, co by postava vyšla ze záběru. Aktuální pozici kamery v Mayavi poskytuje funkce `mlab.move()`. Vrací n-tici obsahující pozici kamery a ohniskový bod. Pro aktualizaci pozice kamery slouží stejná funkce, přičemž očekává tři argumenty, které reprezentují *relativní* posun kamery. Pseudokód pohybu kamery je znázorněn algoritmem 4.4.

Obrázek 4.4: Pseudokód pro pohyb virtuální kamerou

```
# Pohyb postavy
motion = ...

# Získání počáteční pozice krku a~kamery
neck = motion[0]["neck"]
cam, _ = mlab.move()
to_cam = neck - cam

for position in motion:
    # Aktualizace kostry
    ...

    # Výpočet pohybu kamery
    neck = position["neck"]
    cam, _ = mlab.move()
    move = cam - neck + to_cam

    # Pohyb kamery přesáhnul vzdálenost, pozice kamery se aktualizuje
    if np.linalg.norm(move) > treshold:
        x, y, _ = move * multiplier
        mlab.move([x, y, 0])
```

Samozřejmě by se daly vymyslet komplikovanější způsoby, které by mohly zahrnovat rotaci kamery a podobně, ale to není předmětem této práce. Relativní pohyby kamery jsou uloženy do seznamu pro pozdější použití pro aplikování Motion LoRA.

4.4 Generování videa

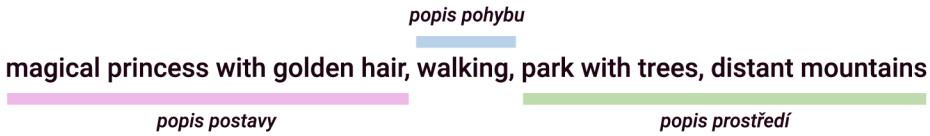
Implementace generování videa byla provedena pomocí knihovny Diffusers[15]. Diffusers je knihovna implementující různorodé difúzní modely pro generování obrázků, videí, zvuku a dokonce i 3D struktur molekul. Jedná se o řešení pro inferenci i trénování vlastních difúzních modelů. Knihovna je navržena s důrazem na principu použitelnosti před výkonem. Tato část byla částečně převzata z dokumentace Diffusers[15].

Pro generování videa je použitá upravená komunitní pipeline `pipeline_animatediff_controlnet.py`. Tato pipeline umožňuje využívat adaptér AnimateDiff v kombinaci s ControlNet sítí. Incializace komunitní pipeline se v knihovně Diffusers provádí předáním argumentu `custom_pipeline` metodě `DiffusionPipeline.from_pretrained`. Výstupem tohoto kroku je video s rozlišením 912×512 pixelů a snímkovou frekvencí 10 snímků za sekundu. Implementace generování videa se nachází v souboru `diffusion.py`.

4.4.1 Manažer embeddingů

Manažer embeddingů (anglicky *embed manager*) je má implementace pro postupnou změnu jednotlivých embeddingů v rámci jedné scény. Jedná se o třídu, která zpřístupňuje tři metody:

Metoda `precompute_embeds` Metoda předpočítává z textových vstupů dané embeddingy. Textovými vstupy jsou popis postavy, popis pohybu postavy a popis vzhledu prostředí. Ke každé trojici vstupů je navíc vložená informace, do jakého snímku platí. Tyto vstupy jsou pomocí knihovny Compel[9] zkombinovány do jednoho embeddingu. Kombinace jednotlivých popisů do jednoho promptu je znázorněna na obrázku 4.5. Seznam těchto embeddingů je následně uložen do instance třídy spolu s příslušnými časovými značkami.



Obrázek 4.5: Znázornění skládání promptu

Metoda `get_embed_window` Metoda `get_embed_window` má argumenty start a stop, které definují okno, pro které se má vrátit tensor reprezentující embedding.

Metoda `get_negative_embeds` Metoda `get_negative_embeds` vrací tensor pro negativní embedding. Ve vychozím nastavení je použit prompt „worst quality, low quality“.

4.4.2 Úpravy komunitní pipeline

Komunitní pipeline bylo pro mé účely nutné mírně upravit. Níže jsou popsány 3 úpravy, které byly provedeny.

[9] https://github.com/damian0815/compel

Počáteční latentní šum V základní implementaci pipeline je počáteční šum videa generován zcela náhodně. Pro zlepšení konzistence bylo generování šumu upraveno technikou FreeNoise jak je popsáno v sekci 2.4. Implementace vychází z oficiální FreeNoise AnimateDiff implementace [16][10].

Batch size Batch size určuje kolik videí, nebo obrázků se má generovat naráz. Obvykle se batch size automaticky detekuje z rozměru tensoru `prompt_embeds`, případně velikosti seznamu `prompts`. To je ale v tomto projektu nežádoucí, protože zde argument `prompt_embeds` vyjadřuje embedding pro každý snímek. Z tohoto důvodu je batch size napevno nastavený na 1.

Krokování Pro možnost delších videí je nutné přidat do pipeline další argumenty. Konkrétně jsem se rozhodl přidat argument `steps_offset`, který říká, kolik kroků odšumnění už bylo v daném latentním prostoru provedeno. Druhým přidaným argumentem je `do_inference_steps`, který značí, kolik kroků odšumnění se má provést. V normálním případě odšumnění začíná v kroku 0 až do kroku daného argumentem `num_inference_steps`. Využití této funkcionality bude vysvětleno v následující sekci.

4.4.3 Využití IP adaptéru

Za účelem zlepšení konzistence postav je využitý IP adaptér. Pro každou scénu je vygenerován samostatným Stable Diffusion modelem obrázek postavy. Obrázek je následně použit pro IP adaptér, který kontroluje styl. Tento IP adaptér je načten do hlavní pipeline, která generuje video.

4.4.4 Generování dlouhých videí

Knihovna Diffusers nemá v základu implementovanou techniku FreeNoise, proto jsem jí musel doimplementovat. Implementace opět vychází z oficiální FreeNoise AnimateDiff implementace [16]. Změny se nacházejí v souboru `unet_3d_blocks.py`, který implementuje U-Net pro kombinaci difúzního modelu Stable Diffusion a pohybového adaptéru AnimateDiff. Konkrétně byly volání funkce `motion_module` nahrazeny voláním vlastní funkce `chunked_motion_module`. Funkce `chunked_motion_module` implementuje samotnou metodu FreeNoise a volá původní `motion_module`. Zbytek sítě U-Net dokáže v základu zpracovat více než 16 snímků, takže tato změna pro implementaci FreeNoise stačí.

S touto základní implementací je při rozlišení 912×512 pixelů možné na grafické kartě s 24 GB video paměti (VRAM) vygenerovat necelých 64 snímků. 64 snímků je výrazně lepší než 16, ale více by bylo ještě lepší. Naštěstí je poměrně jednoduché značně snížit paměťové nároky. Toho je možné dosáhnout sekvenčním zpracováním latentního prostoru namísto zpracování celého prostoru naráz. Jednodušší částí bylo přepnout bloky typu transformer na sekvenční zpracování. Tato funkcionalita je v knihovně Diffusers implementována, avšak je potřeba jí aktivovat. Pro účely tohoto projektu jsem se rozhodl sekvenční zpracování aktivovat pro všechny transformer bloky. Správným nastavením `self._chunk_size` a `self._chunk_dim` v konstruktorech tříd modulu `attention.py` je dosaženo globálního sekvenčního zpracování bloků typu transformer.

Druhým krokem k dosažení nižších paměťových nároků je zapnutí sekvenčního zpracování u bloků `ResnetBlock2D`. Pro bloky typu Resnet není sekvenční zpracování v knihovně

[10] https://github.com/arthur-qiu/FreeNoise-AnimateDiff/

Diffusers implementováno. Toho jsem dosáhl přejmenováním původní metody `forward` a napsáním vlastní metody `forward`, která volá původní přejmenovanou metodu. Nová metoda nejprve rozděluje vstupní tensory na jednotlivé snímky pomocí metody `torch.chunk` a následně volá původní metodu se samostatnými snímky. Návratové hodnoty jednotlivých volání jsou zpět spojeny do jednoho tensoru pomocí metody `torch.cat` a tento tensor je navrácen. Implementace je inspirována implementací v souboru `attention.py` a nachází se v souboru `resnet.py`.

Díky těmto úpravám je možné na stejné grafické kartě generovat více než 128 souvislých snímků videa. Pokud je potřeba delší scéna, je implementována technika použitá v [9]. Technika spočívá v rozdělení velkého latentního prostoru na pevné segmenty, které se vzájemně překrývají o několik snímků. Tato okna jsou postupně o jeden krok odšumněna a poté se překrývající se části výsledků aritmeticky zprůměrují. Tento postup je zopakován několikrát podle požadovaného počtu kroků odšumnění. Technika je popsaná algoritmem 1.

Algoritmus 1: Algoritmus prodlužování videa

```
latents ← rand();    // (batch size, kanály, počet snímků, výška, šířka)
for i ← 0 to num_inference_steps do
    count ← [0] * num_frames;
    accumulated ← zeros_like(latents);
    for window do
        window_latents ← latents[:, :, window, :, :];
        new_latents ← infer(window_latents);
        accumulated[:, :, window, :, :] ← accumulated[:, :, window, :, :]+new_latents;
        count[window] ← count[window] + 1;
    end
    latents ← accumulated/count;
end
```

4.4.5 Aplikování Motion LoRA

Při pohybu virtuální kamery je záhodno, aby se prostředí náležitě hýbalo. Za tímto účelem byly vybrány tyto 4 Motion LoRA:

- `guoyww/animatediff-motion-lora-pan-left` Sloužící pro pohyb kamery doleva
- `guoyww/animatediff-motion-lora-pan-right` Sloužící pro pohyb kamery doprava
- `guoyww/animatediff-motion-lora-tilt-down` Sloužící pro pohyb kamery dolů
- `guoyww/animatediff-motion-lora-tilt-up` Sloužící pro pohyb kamery nahoru

V knihovně Diffusers jsou nejprve tyto Motion LoRA načteny pomocí `pipe.load_lora_weights`. Následně jsou všechny adaptéry vypnuty pomocí `pipe.set_adapters([])`. V základu knihovna Diffusers dokáže aktivovat adaptéry staticky, to znamená, že na celé video je použitý jeden pohyb. Kamera se ale hýbe dynamicky a proto je třeba aktivovat dynamicky i adaptéry. Toho bylo dosaženo aktivováním příslušných adaptéru ve výše popsaném `chunked_motion_module`. Pro každé okno 16 snímků je vypočítán průměrný pohyb kamery a na základě toho jsou aktivovány příslušné adaptéry. Jak bylo dříve zmíněno, adaptéry byly natrénovány na datech s vodoznaky, proto je použita váha 0.7. Zvolená váha je

dobrý kompromis mezi dostatečným pohybem a minimálními artefakty. Vytvořený postup je znázorněný pseudokódem 4.6.

Obrázek 4.6: Pseudokód aplikovaní AnimateDiff Motion LoRA

```
def calculate_active_loras(camera_motions, t_start, t_end):
    # Získání aktuálního okna
    window = torch.tensor(camera_motions[t_start:t_end])

    # Aritmetický průměr pohybů kamery přes celé okno
    avg_motion = torch.mean(window, dim=0)

    # Pokud průměrný pohyb přesáhne práh, je aktivována příslušná LoRA
    loras = []
    if -avg_motion[0] > threshold_horizontal:
        loras.append("left")
    if avg_motion[0] > threshold_horizontal:
        loras.append("right")
    if avg_motion[1] > threshold_vertical:
        loras.append("up")
    if -avg_motion[1] > threshold_vertical:
        loras.append("down")
    return loras
```

4.5 Zvýšení rozlišení videa

Výstupní video z předchozího kroku má rozlišení $912 \times 512 \sim 16 : 9$. To je pro finální video nedostačující, proto je v této práci použit Real-ESRGAN[23]. Real-ESRGAN používá architekturu GAN (*Generative adversarial network*), kde dvě neuronové sítě soutěží proti sobě. Jedna generuje obsah a druhá se snaží rozpoznat, zda je obsah vygenerován, nebo je pravý. Architektura GAN lze použít pro generování obrázků, podobně jako difúzní modely, ale v této práci je použita pouze pro zvýšení rozlišení finálního videa. Za účelem zvýšení rozlišení obrázků nebo videa lze použít i difúzní modely. Tato varianta nebyla zvolena z důvodu velmi malé rychlosti oproti sítím GAN.

Implementace využívá Python knihovnu `basicsr`, která tuto funkcionalitu poskytuje. Konkrétně je použit předtrénovaný model `realesr-animevideov3`[11]. Tento model je přímo určený pro zvětšování rozlišení u animovaných videí. Moje implementace ho provozuje v režimu 4×. To znamená, že z videa o rozlišení 912×512 pixelů se stává video s rozlišením 3648×2048 pixelů, které je následně zmenšeno na standardní Full HD rozlišení 1920×1080 pixelů. Režim 3× by vyprodukoval video s rozlišením 2736×1536, což vypadá po zmenšení na rozlišení Full HD hůře, proto tento režim nebyl zvolen. Ukázka zvětšení je na obrázku 4.7. Tato funkcionalita je implementována v souboru `upscale.py`.

[11] Přehled předtrénovaných Real-ESRGAN modelů se nachází na adrese https://github.com/xinntao/Real-ESRGAN/blob/master/docs/model_zoo.md

Obrázek 4.7: Ukázka zvýšení rozlišení pomocí techniky Real-ESRGAN. Původní obrázek vlevo, zvětšený vpravo. Obrázky jsou převzaty z [23] (Upraveno).

4.6 Generování scén bez postavy

Jako bonus bylo implementováno generování scén bez postavy. To může sloužit pro tvoření vyplňujících záběrů, například pohled na samotný les. Z uživatelského hlediska je toho dosaženo vynecháním popisu postavy a popisu pohybu. Scény s pohybem a bez pohybu lze libovolně prokládat.

Po implementační stránce funguje generování podobně, jako generování záběrů s postavou. Hlavním rozdílem je, že se samozřejmě negeneruje pohyb postavy. Namísto toho se nastaví váha IP adaptéru a sítě ControlNet na 0. Zároveň se vytvoří statický pohled kamery, aby se neaplikovaly pohybové LoRA adaptéry. Tyto změny zajišťují generování scén bez postavy.

4.7 Uživatelské rozhraní

Uživatelské rozhraní je implementováno za pomocí knihovny Gradio. Gradio je knihovna pro Python, která umožňuje rychle vytvořit webovou aplikaci pro modely strojového učení. Pomocí vestavěné funkce je možné webovou aplikaci zdarma sdílet do internetu bez nutnosti veřejné IP adresy. Veškerá implementace rozhraní je v jazyce Python. Tento text byl převzat z [1].

Rozhraní je budováno pomocí deklarativního přístupu. To znamená, že je pouze definováno umístění prvků a jejich propojení. O interaktivitu se stará knihovna Gradio na pozadí.

Uživatelského rozhraní umožňuje do textového pole psát vstupní příběh. Po kliknutí na tlačítko „Generate scenario“ je pomocí velkého jazykového modelu vygenerován scénář. Tento scénář je následně možné libovolně editovat.

Dále je možné v sekci „Advanced settings“ nastavovat parametry generování. To zahrnuje nastavení počtu kroků, použité modely a podobně. Pomocí přepínače `--demo` lze toto nastavení skrýt. Tento přepínač se především hodí v uživatelské studii — aby uživatelé testovali nástroj se stejnými modely a výsledky se daly porovnat.

Po dokončení scénáře a případného nastavení je možné stiskem tlačítka „Generate video“ zahájit generování videa. Po vygenerování videa je výsledné video spolu s řídícími snímky prezentováno na pravé straně uživatelského rozhraní. Vytvořené uživatelské rozhraní za pomocí knihovny Gradio je na obrázku 4.8.

4.7.1 Prvek scénáře

Knihovna Gradio poskytuje základní stavební prvky pro budování uživatelského rozhraní. Těmi jsou například textová pole, tlačítka, grafy, zobrazovače videí a podobně. Editovat JSON scénář v textovém poli je sice funkční možnost, avšak uživatelsky velmi nepřívětivé. V některých případech je lepší vytvořit nový prvek, což je případ této práce. Vlastní prvek lze vytvořit příkazem `gradio cc create MyComponent`.

Celá implementace vlastního editoru scénáře se nachází ve složce `gradio_scenario`. Dělí se na dvě části:

- Frontend
- Backend

Frontend implementace vlastních prvků je v programovacím jazyce Typescript za pomocí reaktivní knihovny Svelte, nachází se ve složce `frontend`. Backend implementace je v programovacím jazyce Python, nachází se ve složce `backend`.

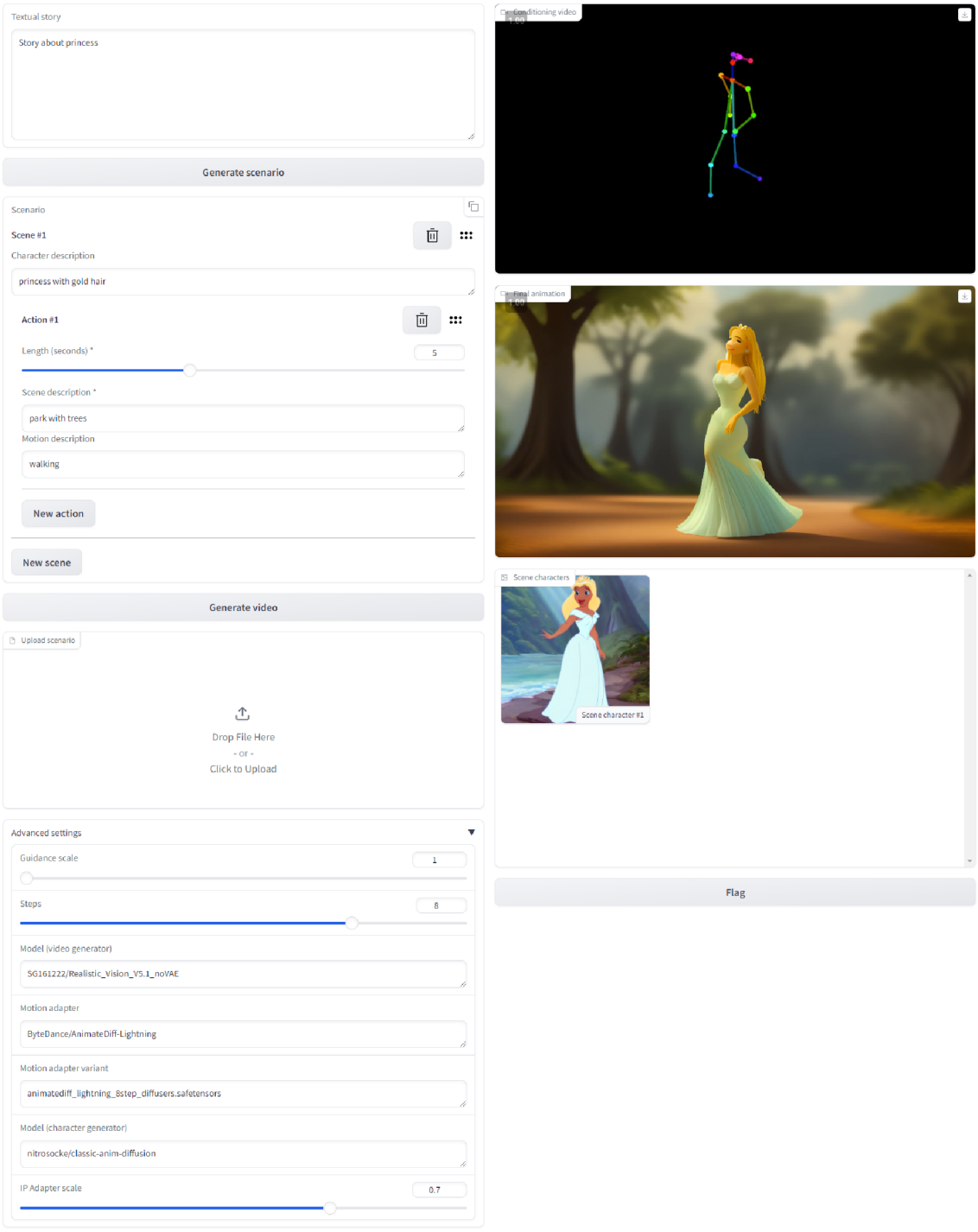
Prvek scénáře umožňuje uživatelsky přívětivou editaci scénáře popsaného výše. Umožňuje dynamicky tvořit a mazat scény, stejně tak je možné tvořit a mazat příslušné akce. Další implementovanou funkcionalitou je možnost přesouvání pořadí scén a akcí pomocí funkcionality drag-and-drop. Funkcionalita přesouvání myší byla implementována pomocí knihovny `svelte-dnd-action`[12]. Dále je možné stáhnout vlastní JSON scénář tlačítkem v pravém horním rohu. Hlavní část zdrojových kódů scénáře je v souboru `Index.svelte`.

Účel backendu je napojit frontend na zbytek systému. Obsahuje dvě metody: `preprocess` a `postprocess`. Metoda `postprocess` má obecně za úkol transformovat vstupní data z jazyka Python do dat čitelných pomocí jazyka Javascript (Typescript). V mé implementaci to znamená transformaci z Python `dict` do JSON objektu. Při této transformaci navíc probíhá přidání unikátních identifikátorů scén a akcí, potřebných pro frontendovou část. Metoda `preprocess` má přesně opačnou úlohu. Data z frontendu transformuje zpět do Python reprezentace. Při této transformaci jsou naopak pomocné identifikátory odebrány.

4.7.2 Validace vstupů

Vstupy jsou v Gradio rozhraní řádně validovány. Na chyby je uživatel upozorněn pomocí vyskakovacích oken. Popisy chyb jsou koncipovány tak, aby byl uživatel schopen provést korekci autonomně.

[12] https://github.com/isaacHagoel/svelte-dnd-action



Obrázek 4.8: Vytvořené uživatelské rozhraní pro tvoření animovaných příběhů

Kapitola 5

Vyhodnocení a uživatelská studie

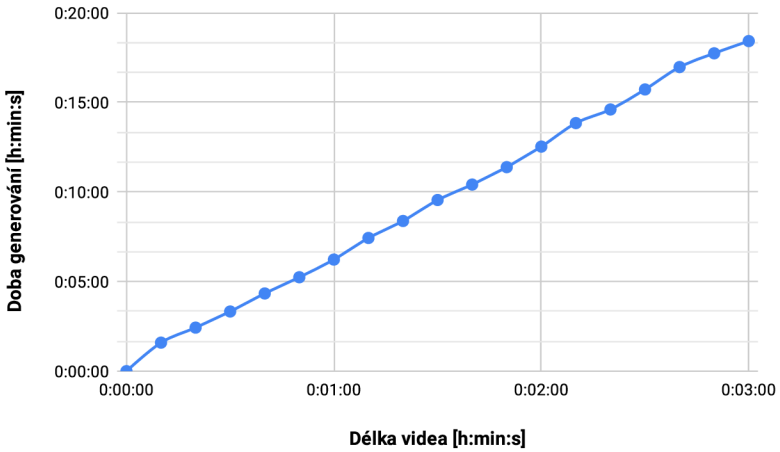
Tato kapitola se zaměřuje na vyhodnocení implementace autorem, sběr dat pomocí uživatelské studie a její následné vyhodnocení. Uživatelská studie je dle zadání zaměřená na konzistenci ztvárnění postav, struktury vyprávění a intuitivnost ovládání. Veškeré vyhodnocování bylo provedeno s 8 krokovou variantou pohybového adaptéru AnimateDiff lightning. Výchozí model generující video je `SG161222/Realistic_Vision_V5.1_noVAE`, výchozí model generující vzhled postav je `nitrosocke/classic-anim-diffusion`. Tyto modely byly použity pro uživatelskou studii.

5.1 Vyhodnocení implementace

V této sekci provedu vlastní zhodnocení implementace. Jedná se primárně o vlastní postřehy.

5.1.1 Rychlost generování

Byla dosažena plánovaná rychlost generování. Jednomitové video se generuje přibližně 6 minut. Doba generování vůči délce videa je znázorněna na obrázku 5.1. Z grafu lze vyčíst, že systém se škáluje přibližně lineárně.



Obrázek 5.1: Graf rychlosti generování videa na grafické kartě Nvidia GeForce RTX 4090

5.1.2 Dosažená konzistence

Dosažená konzistence se hodně odvíjí od použitého latentního difúzního modelu generující video. Některé modely s pohybovým adaptérem fungují skvěle, jiné příšerně. Například model navrhovaný v zadání práce `nitrosocke/classic-anim-diffusion` produkuje velmi špatné výsledky. Naproti tomu model `SG161222/Realistic_Vision_V5.1_noVAE` funguje velmi dobře. Porovnání výsledků těchto dvou modelů je na obrázku 5.2.

Obrázek 5.2: Porovnání výsledného řešení s různými modely — *nitrosocke/classic-anim-diffusion* vlevo, *SG161222/Realistic_Vision_V5.1_noVAE* vpravo

U některých snímků je špatný počet končetin, deformované končetiny a další jiné problémy. Výrazně trpí obličeje, zejména pokud je postava daleko. Ukázky těchto problémů jsou vyobrazeny na obrázku 5.3. Tyto problémy by se daly redukovat použitím lepšího difúzního modelu.

Obrázek 5.3: Příklady nedostatků difúzního latentního modelu Stable Diffusion 1.5

5.2 Průběh sběru dat

Sběr dat probíhal pomocí Google formuláře a Gradio prostředí. Uživatelé, kteří byli součástí šetření, nepotřebovali žádné zvláštní dřívější znalosti. Podmínkou byl pouze počítač s funkčním moderním webovým prohlížečem a připojením k internetu.

V prvním kroku byli uživatele seznámeni s principem a fungováním sběru dat. Toto seznámení proběhlo prostřednictvím první strany Google formuláře. Seznámení zahrnovalo vysvětlení ovládání uživatelského rozhraní, seznámení s omezeními systému a pokyny pro následnou zpětnou vazbu.

V druhém kroku si uživatelé v připraveném demonstračním prostředí Gradio zkusili vygenerovat scénář na základě textového příběhu. Následně mohli tento scénář libovolně upravit. Když byli spokojeni se svým detailním scénářem, vygenerovali finální animaci. Po vygenerování animace kliknutím na tlačítko „Flag“ volitelně označili data k další analýze. Již zmiňované rozhraní je vyobrazeno na obrázku 4.8.

Ve třetím kroku vyplnili Google formulář pro předání zpětné vazby. V otázkách s bodovým rozsahem se hodnotilo jako ve škole — 1 je nejlepší, 5 znamená nejhorší. Formulář obsahoval následující body:

1. Zhodnoťte konzistenci ztvárnění postavy — zda postava zůstává v průběhu času stejná.
   (1, Postava je zcela stejná – 5, Postava se mění k nerozeznání)

2. Zhodnoťte, jak moc se postava podobá jejímu textovému popisu.
   (1, Postava se zcela podobá jejímu popisu – 5, Postava je zcela odlišená od jejího popisu)

3. Zhodnoťte konzistenci prostředí — zda prostředí zůstává v průběhu času stejné.
   (1, Prostředí zůstává stejné – 5, Prostředí se mění k nerozeznání)

4. Zhodnoťte, jak moc se prostředí podobá jeho textovému popisu.
   (1, Prostředí se podobá jeho textovému popisu – 5, Prostředí se vůbec nepodobá jeho popisu)

5. Zhodnoťte intuitivnost ovládání uživatelského rozhraní.
   (1, Vše mi bylo jasné – 5, Vůbec se v tom nedokážu orientovat)

6. Zhodnoťte funkčnost generování scénáře z textového popisu.
   (1, Scénář nebylo prakticky třeba upravovat – 5, Zcela nefunkční)

7. Jaký je váš celkový dojem z tohoto nástroje?
   (1, Skvělý – 5, Hrozný)

8. Zde můžete napsat slovní doplnění k bodům výše.
   (Volitelné)

9. Napište, co byste zlepšili.
   (Volitelné)

5.3 Vyhodnocení uživatelské studie

Uživatelské studie se celkem zúčastnilo 10 respondentů. Uživatelé byli osloveni osobně a pomocí platformy Discord. Číselné hodnocení otázek je shrnuto v tabulce 5.1.

Tabulka 5.1: Numerické výsledky uživatelské studie

Otázka	**Hodnocení**	
	Průměrné	**Medián**
1. — Konzistence postav	2.8	2
2. — Podoba postav vůči jejich popisu	1.8	2
3. — Konzistence prostředí	3.0	3
4. — Podoba prostředí vůči jejich popisu	2.6	3
5. — Intuitivnost ovládání UI	1.6	2
6. — Generování scénáře	1.8	2
7. — Celkový dojem	1.8	2

Z výsledků lze usoudit, že nástroj dosahuje uspokojivých výsledků. Provedl jsem T-test mezi hodnocením konzistence postav a prostředí, mezi hodnocením podoby postav a prostředí vůči příslušným popisům. Oba testy dosáhly p-hodnoty nad 0.05, takže není dostatečný důkaz, že by se hodnocení výrazně lišila. Při větším vzorku uživatelů by se mohl ukázat opak, proto zde spekulativně nastíním, proč by mohl rozdíl vzniknout.

Konzistence postav je hodnocena lépe, než konzistence prostředí. To přisuzuji použití IP adaptéru pro zlepšení konzistence vzhledu postav. U prostředí stejná metoda nelze jednoduše použít, by bylo zapotřebí vymyslet jiný způsob.

Podoba postav vůči jejich popisu je též hodnocena lépe, než podoba prostředí vůči jejich popisu. Za to pravděpodobně mohou tři faktory:

- IP adaptér pro vzhled postav
- Popis postavy je v textovém promptu dříve, než popis prostředí
- Vybraný Stable Diffusion model

Vzhled postav se generuje pro IP adaptér zvlášť, z toho důvodu se více podobá popisu, protože není zatížený popisem prostředí. Z mého pozorování jsou popisy na začátku promptu více zohledněny ve finálním obrázku. Kvůli tomu se prostředí méně podobá jeho popisu. Další pravděpodobnou možností k vysvětlení tohoto výsledku je použitý model. Použitý model je pravděpodobně primárně naučený na postavách a prostředí je zohledněno méně.

Uživatelé bohužel příliš nechtěli odpovídat na volné otázky, s výjimkou dvou uživatelů, kteří odpověděli na otázku číslo 8:

> „Zajímavý nástroj, každopádně má ještě hodně daleko k dokonalosti“

> „Je to pecka!“

Odpovědi dle mého výstižně reflektují průměrné bodové hodnocení respondentů. Výsledky souhlasí s mým osobním dojmem.

Ač není uživatelská studie zaměřená na testování, byla díky ní odhalena jedna chyba. Díky sběru dat pomocí tlačítka „Flag“ jsem mohl chybu jednoduše zreprodukovat za účelem opravy. Chyba se projevovala při generování akcí kratších než 4 sekundy. Tato chyba je ve finální implementaci opravena.

5.3.1 Možná zlepšení

Tato sekce krátce rozebírá možná zlepšení s ohledem na definovaná omezení. Možnosti rozšíření jsou nastíněny v závěru práce.

Konzistence a ztvárnění postav by se dala zlepšit pomocí novějších modelů. Konkrétně pomocí kombinace Stable Diffusion XL a adaptéru AnimateDiff. AnimateDiff adaptér bohužel zatím není v dobrém stavu pro použití s modelem Stable Diffusion XL. Navíc ControlNet sítě s modelem Stable Diffusion XL nejsou na stejné úrovni jako u modelu Stable Diffusion 1.5. Zbývá tak počkat na zlepšení podpory, případně je možnost počkat, co přinese model Stable Diffusion 3.

Intuitivnost uživatelského rozhraní bych zlepšil na základě výstupů z dalšího detailního dotazníku, který by byl zaměřený přímo na tento aspekt. Momentálně podle uživatelské studie dosahuje dobrého hodnocení, takže bych se primárně zaměřil na ostatní témata.

Generování scénáře by šlo vylepšit využitím většího jazykového modelu, případně zlepšením promptu. Další možností by bylo dotrénování existujícího jazykového modelu přímo za účelem generování scénáře z textového popisu.

Kapitola 6

Závěr

Cílem této práce bylo vytvořit program umožňující automatické generování animovaných příběhů z textového popisu. Tento cíl se úspěšně podařilo splnit.

Nejdříve jsem se seznámil s různými metodami generování animovaných videí a pohybů lidské kostry. Tyto metody se primárně skládají z difúzních modelů, o kterých jsem vytvořil stručný přehled. Řadu těchto modelů jsem vyzkoušel a následně vybral ty nejvhodnější pro finální implementaci. Implementoval jsem systém, který spojuje difúzní modely generující animovaná videa a pohyb lidské postavy pro dosažení konzistentních animovaných příběhů o jedné postavě v dané scéně. Systém vygeneruje jednominutové video za přibližně 6 minut. Za účelem intuitivního ovládání jsem vytvořil uživatelské rozhraní pomocí nástroje Gradio[1]. Systém byl následně otestován v uživatelské studii s přijatelnými výsledky. Na závěr jsem vytvořil plakát, který prezentuje práci jako celek.

Uživatelská studie byla provedena na základě bodového hodnocení, při kterém se známkovalo jako ve škole. Průměrné hodnocení konzistence postav je 2.8 bodů, průměrné hodnocení konzistence prostředí je 3.0 bodů. Intuitivnost uživatelského rozhraní je 1.6 bodů. Toto hodnocení naznačuje funkční systém, ale zároveň z něj lze odvodit, že je zde ještě velký prostor pro zlepšení.

V práci bych chtěl pokračovat tak, že bych použil nové modely, které vznikly před odevzdáním této práce. To by umožnilo lepší vzhledovou konzistenci, lepší ztvárnění a kvalitnější generování pohybu. Další rozšířením, které bych rád implementoval, by bylo přidání mluveného slova a titulků. Poslední změnou, kterou bych rád implementoval je možnost granuálně editovat scény. Aktuálně je třeba v systému pro změnu jedné scény celé video generovat znovu. Tyto rozšíření nebyly implementovány z důvodu zaměření pozornosti na hlavní body práce.

Někdo další by mohl pokračovat odstraněním hlavního omezení, kterým je maximálně jedna postava ve scéně. Toho by se možná dalo dosáhnout kombinací technik z [20]. Tento problém je velmi komplexní a nekončí u generování pohybu. Generování dvou postav v jedné scéně s konzistentním vhledem je mnohonásobně těžší problém, který vyžaduje překonat velké výzvy. Možným směrem je generování samotných postav bez pozadí pomocí nové techniky LayerDiffuse[27] a následná kompozice do jednoho snímku. Další možností je použít techniku s názvem Regional Prompter[1], která umožňuje zvlášť specifikovat vzhled jednotlivých částí obrázku. Zde by se však nastal problém při překrývajících se postavách.

[1] https://github.com/hako-mikan/sd-webui-regional-prompter

Literatura

[1] Abid, A.; Abdalla, A.; Abid, A.; Khan, D.; Alfozan, A. et al. *Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild.* arXiv, 2019. Dostupné z: https://arxiv.org/abs/1906.02569.

[2] Blattmann, A.; Dockhorn, T.; Kulal, S.; Mendelevitch, D.; Kilian, M. et al. *Stable Video Diffusion: Scaling Latent Video Diffusion Models to Large Datasets.* arXiv, 2023. Dostupné z: https://arxiv.org/abs/2311.15127.

[3] Cao, Z.; Hidalgo, G.; Simon, T.; Wei, S.-E. a Sheikh, Y. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.* arXiv, 2018. Dostupné z: https://arxiv.org/abs/1812.08008.

[4] Chintan, P. Stable Diffusion Explained: How Text Prompts Become Stunning Images. *Medium* online. 5. března 2024. Dostupné z: https://medium.com/@patelchintan732000123/a-comprehensive-look-at-text-encoders-vq-gans-and-u-nets-3b4687bc8b58. [vid. 9. 4. 2024].

[5] Cooper, B. BMWalker to Open Pose. *Observable* online. 10. června 2023. Revidováno 10. 6. 2023. Dostupné z: https://observablehq.com/@hellonearthis/bwalker-to-open-pose. [vid. 9. 4. 2024].

[6] Guo, Y.; Yang, C.; Rao, A.; Liang, Z.; Wang, Y. et al. *AnimateDiff: Animate Your Personalized Text-to-Image Diffusion Models without Specific Tuning.* arXiv, 2023. Dostupné z: https://arxiv.org/abs/2307.04725.

[7] Hu, E. J.; Shen, Y.; Wallis, P.; Allen Zhu, Z.; Li, Y. et al. *LoRA: Low-Rank Adaptation of Large Language Models.* arXiv, 2021. Dostupné z: https://arxiv.org/abs/2106.09685.

[8] Khachatryan, L.; Movsisyan, A.; Tadevosyan, V.; Henschel, R.; Wang, Z. et al. *Text2Video-Zero: Text-to-Image Diffusion Models are Zero-Shot Video Generators.* arXiv, 2023. Dostupné z: https://arxiv.org/abs/2303.13439.

[9] Kosinski, J. *Improved AnimateDiff for ComfyUI and Advanced Sampling Support* online. GitHub, 2023. Dostupné z: https://github.com/Kosinkadink/ComfyUI-AnimateDiff-Evolved. [vid. 15. 4. 2024].

[10] Lin, S. a Yang, X. *AnimateDiff-Lightning: Cross-Model Diffusion Distillation.* arXiv, 2024. Dostupné z: https://arxiv.org/abs/2403.12706.

[11] Loper, M.; Mahmood, N.; Romero, J.; Pons Moll, G. a Black, M. J. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia).* ACM, říjen 2015, sv. 34, č. 6, s. 248:1–248:16.

[12] Luo, S.; Tan, Y.; Patil, S.; Gu, D.; Platen, P. von et al. *LCM-LoRA: A Universal Stable-Diffusion Acceleration Module*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2311.05556.

[13] Mangrulkar, S.; Gugger, S.; Debut, L.; Belkada, Y.; Paul, S. et al. *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods* https://github.com/huggingface/peft. 2022.

[14] Mou, C.; Wang, X.; Xie, L.; Wu, Y.; Zhang, J. et al. *T2I-Adapter: Learning Adapters to Dig out More Controllable Ability for Text-to-Image Diffusion Models*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2302.08453.

[15] Platen, P. von; Patil, S.; Lozhkov, A.; Cuenca, P.; Lambert, N. et al. *Diffusers: State-of-the-art diffusion models* https://github.com/huggingface/diffusers. GitHub, 2022.

[16] Qiu, H.; Xia, M.; Zhang, Y.; He, Y.; Wang, X. et al. *FreeNoise: Tuning-Free Longer Video Diffusion via Noise Rescheduling*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2310.15169.

[17] Radford, A.; Kim, J. W.; Hallacy, C.; Ramesh, A.; Goh, G. et al. *Learning Transferable Visual Models From Natural Language Supervision*. arXiv, 2021. Dostupné z: https://arxiv.org/abs/2103.00020.

[18] Ramachandran, P. a Varoquaux, G. Mayavi: 3D Visualization of Scientific Data. *Computing in Science and Engineering*. Institute of Electrical and Electronics Engineers (IEEE), březen 2011, sv. 13, č. 2, s. 40–51. ISSN 1521-9615. Dostupné z: http://dx.doi.org/10.1109/MCSE.2011.35.

[19] Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P. a Ommer, B. *High-Resolution Image Synthesis with Latent Diffusion Models*. arXiv, 2021. Dostupné z: https://arxiv.org/abs/2112.10752.

[20] Shafir, Y.; Tevet, G.; Kapon, R. a Bermano, A. H. *Human Motion Diffusion as a Generative Prior*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2303.01418.

[21] Tevet, G.; Raab, S.; Gordon, B.; Shafir, Y.; Cohen Or, D. et al. *Human Motion Diffusion Model*. arXiv, 2022. Dostupné z: https://arxiv.org/abs/2209.14916.

[22] Wang, J.; Yuan, H.; Chen, D.; Zhang, Y.; Wang, X. et al. *ModelScope Text-to-Video Technical Report*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2308.06571.

[23] Wang, X.; Xie, L.; Dong, C. a Shan, Y. *Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data*. arXiv, 2021. Dostupné z: https://arxiv.org/abs/2107.10833.

[24] Wright, A.; Andrews, H.; Hutton, B. a Dennis, G. *JSON Schema: A Media Type for Describing JSON Documents*. JSON Schema, 2020. Dostupné z: https://json-schema.org/draft/2020-12/json-schema-core.html.

[25] Ye, H.; Zhang, J.; Liu, S.; Han, X. a Yang, W. *IP-Adapter: Text Compatible Image Prompt Adapter for Text-to-Image Diffusion Models*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2308.06721.

[26] Zhang, D. J.; Wu, J. Z.; Liu, J.-W.; Zhao, R.; Ran, L. et al. *Show-1: Marrying Pixel and Latent Diffusion Models for Text-to-Video Generation*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2309.15818.

[27] Zhang, L. a Agrawala, M. *Transparent Image Layer Diffusion using Latent Transparency*. arXiv, 2024. Dostupné z: https://arxiv.org/abs/2402.17113.

[28] Zhang, L.; Rao, A. a Agrawala, M. *Adding Conditional Control to Text-to-Image Diffusion Models*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2302.05543.

[29] Zhang, S.; Wang, J.; Zhang, Y.; Zhao, K.; Yuan, H. et al. *I2VGen-XL: High-Quality Image-to-Video Synthesis via Cascaded Diffusion Models*. arXiv, 2023. Dostupné z: https://arxiv.org/abs/2311.04145.

[30] Zucconi, A. *An Introduction to Procedural Animations* online. 17. dubna 2017. Dostupné z: https://www.alanzucconi.com/2017/04/17/procedural-animations/. [vid. 22. 4. 2024].

Příloha A

Obsah paměťového média

V této příloze je popsán obsah odevzdaného paměťového média.

- `/` ..... Paměťové médium
  - `latex/` ..... Zdrojové soubory textu ve formátu LaTeXa výsledné PDF
  - `src/` ..... Zdrojové kódy práce
    - `gradio_scenario/` ..... Vlastní Gradio komponenta pro scénář
    - `diffusers/` ..... Upravená knihovna Diffusers
    - `priorMDM/` ..... Upravený repozitář priorMDM
    - `patches/` ..... Shrnutí úprav existujících kódů
    - `models/` ..... Malé modely, které se automaticky nestahují
    - `script/` ..... Pomocné scripty pro vytvoření prostředí
    - `*.py` ..... Hlavní zdrojové soubory projektu
  - `examples/` ..... Příklady vygenerovaných videí
  - `README.md` ..... Popis paměťového média
  - `plakat.pdf` ..... Plakát

Příloha B

Plakát

AUTOMATICKÁ TVORBA ANIMOVANÉHO VIDEA NA ZÁKLADĚ TEXTOVÉHO PŘÍBĚHU

AUTOR: JOSEF KUCHAŘ VEDOUCÍ: DOC. RNDR. PAVEL SMRŽ, PH.D. 2024

PRÁCE

Cílem práce bylo vytvořit systém pro automatické generování animovaného videa z textového popisu. Toho bylo dosaženo pomocí dvou difúzních modelů a několika pomocných adaptérů pro udržení konzistence. Jeden difúzní model generuje pohyb postavy, druhý difúzní model generuje samotné video.

Program je napsaný v programovacím jazyce Python. Implementace generování videa byla provedena pomocí knihovny Diffusers. Pohyb postavy je generován oficiální implementací priorMDM. Intuitivní uživatelské rozhraní bylo vytvořeno pomocí knihovny Gradio. Celé schéma je znázorněno na pravé straně.

Výsledné video má rozlišení 1920x1080 pixelů a snímkovou frekvenci 10 snímků za sekundu.

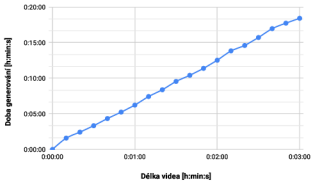
UŽIVATELSKÁ STUDIE

Systém byl otestován v uživatelské studii s přijatelnými výsledky. Uživatelská studie byla provedena na základě bodového hodnocení, při kterém se známkovalo jako ve škole. Výsledky byly reflektovány.

Otázka	Hodnocení Průměrné	Medián
Konzistence postav	2.8	2
Podoba postav vůči jejich popisu	1.8	2
Konzistence prostředí	3.0	3
Podoba prostředí vůči jejich popisu	2.6	3
Intuitivnost ovládání UI	1.6	2
Generování scénáře	1.8	2
Celkový dojem	1.8	2

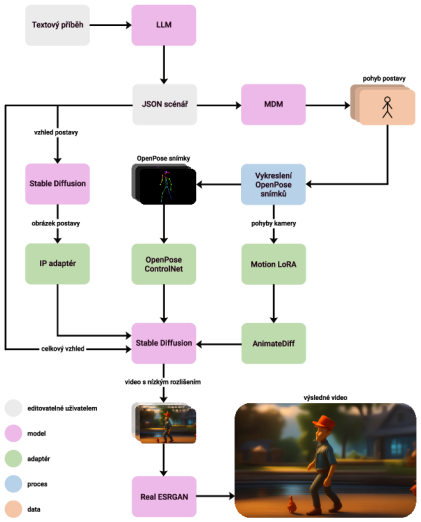
RYCHLOST GENEROVÁNÍ

Program dokáže vygenerovat jednominutové video za přibližně 6 minut. Doba generování se lineárně škáluje s délkou výsledného videa

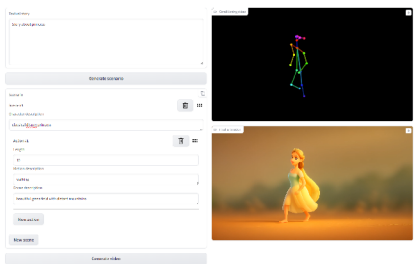


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

SCHÉMA IMPLEMENTACE



UŽIVATELSKÉ ROZHRANÍ



Příloha C

Návod na zprovoznění programu

Program byl testován na fakultních serverech výzkumné skupiny KNOT, ale měl by běžet i na jiných systémech. Podmínkou je grafická karta s alespoň 24 GB grafické paměti. Následující postup předpokládá operační systém Linux. Dále jsou třeba tyto prerekvizity:

- FFmpeg[1]
- Nvidia CUDA[2]

Instalace ostatních nutných knihoven a balíků se provádí automaticky. Veškeré akce probíhají v adresáři `src`. Nejprve je třeba mít nainstalovaný správce balíků Conda[3], případně Miniconda[4]. Na to je připravený script, kterým se Miniconda automaticky nainstaluje:

```
$ chmod +x ./script/01_install_miniconda.sh
$ ./script/01_install_miniconda.sh
```

Poté je třeba vytvořit prostředí, které obsahuje všechny potřebné balíky a knihovny. Toho je dosaženo pomocí scriptu:

```
(base) $ chmod +x ./script/02_setup_environment.sh
(base) $ ./script/02_setup_environment.sh
```

Následně je možné vytvořené prostředí aktivovat:

```
(base) $ conda activate text2video
```

Tímto je příprava dokončena a je možné spustit vytvořený program. Gradio prostředí se spustí pomocí (program vypíše na standardní výstup lokální a sdílenou URL adresu):

```
(text2video) $ python text2video_gradio.py
```

Verze programu, která pracuje v příkazovém řádku se spustí pomocí:

```
(text2video) $ python text2video_cmd.py
```

Popsané argumenty verze programu pracujícího z příkazového řádku se dají získat pomocí příkazu:

```
(text2video) $ python text2video_cmd.py -h
```

[1] https://ffmpeg.org/
[2] https://developer.nvidia.com/cuda-downloads
[3] https://docs.conda.io/
[4] https://docs.anaconda.com/free/miniconda/index.html