

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica - Scienza e Ingegneria (DISI)
Corso di Laurea Magistrale in Ingegneria Informatica

**Generative AI models and frameworks
for philological visualization
in cultural heritage**

**Tesi di Laurea Magistrale in
Fondamenti di Computer Graphics M**

Candidato:
Mariano Mancini

Relatore:
Prof.ssa Serena Morigi

Correlatori:
**Ing. Antonella Guidazzoli
Daniele De Luca**

Anno Accademico 2025-2026

Introduzione

Negli ultimi anni, i modelli generativi basati su intelligenza artificiale hanno registrato un'evoluzione senza precedenti, trovando ampia applicazione in molteplici settori scientifici e umanistici. Il presente lavoro di tesi si inserisce in questo panorama, ponendosi l'obiettivo di coniugare l'innovazione tecnologica con la tutela e la valorizzazione del patrimonio culturale. Nello specifico, la ricerca si è focalizzata sulla ricostruzione di oggetti, elementi architettonici e *avatar* 3D, con un *focus* particolare alla correttezza filologica.

L'attività è stata sviluppata nel quadro dell'iniziativa "Experience Etruria. A Lazio Immersive Digital Exhibition", un progetto di valorizzazione promosso dal Ministero della Cultura e dalla Soprintendenza Archeologia del Lazio e dell'Etruria Meridionale. In collaborazione con il VisitLab CINECA, l'attività ha contribuito alla realizzazione dei contenuti digitali per la mostra immersiva ospitata presso l'Istituto Italiano di Cultura di San Francisco nel maggio 2026.

In questo contesto, la ricerca sull'AI generativa ha permesso di generare immagini e video aventi come soggetto in particolare "Ati", un personaggio etrusco creato precedentemente in computer grafica dal Laboratorio di Visualizzazione del Cineca con le sembianze di una giovane fanciulla etrusca. Unendo rigore scientifico e innovazione tecnologica è stato possibile creare il *video teaser* del progetto. L'attività si è articolata in tre fasi sequenziali e complementari:

- **Fase preliminare: sperimentazione locale e addestramento LoRA.** La fase preliminare si è concentrata sullo studio e analisi di

metodi di addestramento per la creazione di modelli *LoRA* (*Low-Rank Adaptation*) utilizzando lo strumento Kohya_ss. In questa fase, operata su *hardware consumer* utilizzando Stable Diffusion 1.5, sono stati studiati gli elementi necessari e le configurazioni ottimali per garantire la fedeltà del soggetto nelle generazioni statiche.

- **Fase intermedia: migrazione su infrastruttura HPC Leonardo.** Il lavoro è stato traslato sul supercomputer Leonardo per superare i limiti computazionali delle configurazioni locali. Attraverso l'utilizzo di Chappyner, è stato configurato un ambiente ad alte prestazioni per eseguire il *training* e il *testing* di modelli complessi (quali Flux.1, Flux.2 e Z-Image Turbo) all'interno di interfacce avanzate come ComfyUI per il *testing* e musubi-tuner come nuovo strumento di *training* di modelli più recenti.
- **Fase finale: generazione video avanzata e produzione finale.** L'ultima fase ha visto la transizione dall'immagine statica al contenuto dinamico. Tramite una sperimentazione comparativa tra i modelli Wan 2.1, Wan 2.2 e LTX 2, sono state esplorate metodologie di generazione *Image-to-Video*. Questa attività ha portato alla realizzazione di un'opera filmica di circa un minuto, composta da diverse sequenze animate, presentata a San Francisco in occasione di una mostra sull'Etruria sponsorizzata dalla Regione Lazio.

Il presente lavoro di tesi è strutturato in cinque capitoli che guidano il lettore attraverso un percorso lineare, partendo dai fondamenti teorici dell'Intelligenza Artificiale Generativa fino all'implementazione pratica e alla valutazione sperimentale su infrastrutture di calcolo ad alte prestazioni.

- **Capitolo 1: Modelli di Intelligenza Artificiale generativa** definisce il quadro teorico della ricerca. Vengono analizzate l'architettura e le componenti fondamentali dei modelli di diffusione, tracciando l'evoluzione dello stato dell'arte dei modelli *text-to-image*.

- **Capitolo 2: Posizionamento del lavoro svolto** descrive la struttura generale del workflow proposto e le caratteristiche dei modelli *text-to-image* utilizzati in questo lavoro di tesi. Vengono inoltre presentati i principi fondamentali del *fine-tuning* efficiente tramite moduli *LoRA* e l'orchestrazione dei processi mediante l'ecosistema di programmazione visuale *ComfyUI*.
- **Capitolo 3: Pipeline sperimentale e protoipazione locale** descrive la prima fase operativa e metodologica. Viene esaminata l'ingegnerizzazione del *dataset* di alcuni soggetti, dettagliando le tecniche di selezione e *captioning*. Viene poi documentato l'addestramento preliminare eseguito in ambiente locale tramite l'interfaccia *Kohya_ss*, concludendosi con un'analisi quantitativa (funzione di *Loss*) e qualitativa dei primi *output* grafici.
- **Capitolo 4: L'Infrastruttura HPC** introduce il cambio di scala computazionale. Viene illustrato il processo di migrazione dal contesto locale all'ambiente di *High-Performance Computing (HPC)* sul supercomputer Leonardo (CINECA), descrivendo la configurazione dell'ambiente di calcolo e l'evoluzione degli strumenti di *training*, caratterizzati dal passaggio da interfacce grafiche a *script* da linea di comando avanzati.
- **Capitolo 5: Addestramento su Leonardo e Analisi Sperimentale** costituisce il nucleo sperimentale della tesi. Viene approfondito l'addestramento su larga scala in configurazione *Multi-GPU*, presentando un'analisi comparativa del processo di *training* tra i modelli Flux.1, Flux.2 e Z-Image. Il capitolo esamina i test di coerenza visiva effettuati su diversi *sampler* e *scheduler*, definendo infine le metodologie di rifinitura (*Inpainting*) e combinazione simultanea di più moduli *LoRA*.
- **Capitolo 6: Analisi e utilizzo di modelli per generazione video** estende la ricerca alla dimensione temporale. Vengono analizzate le architetture *text-to-video* e *video-to-video* di ultima

generazione (Wan 2.1, Wan 2.2, LTX 2) e la generazione in locale tramite Pinokio. Viene dettagliato il processo di generazione video su Leonardo, la gestione dello spazio latente nel *sampling* video e le tecniche di controllo cinematografico. Il capitolo si chiude con l'integrazione multimediale del progetto, che unisce sintesi vocale adattiva nel video finale.

Infine, la sezione Conclusioni e Sviluppi Futuri traccia un bilancio complessivo della ricerca, mettendo a confronto l'efficienza computazionale dell'approccio locale rispetto a quello *HPC*. Vengono sintetizzati i risultati ottenuti nel delicato equilibrio tra fotorealismo generativo e rigore filologico-archeologico, delineando i limiti tecnologici riscontrati e le prospettive future dell'IA applicata alla museologia digitale.

Sommario

Introduzione	i
Elenco delle figure	x
1 Modelli di Intelligenza Artificiale Generativa	1
1.1 Generative Adversarial Networks (GAN)	2
1.1.1 Limitazioni intrinseche e instabilità matematica .	3
1.2 VAE (Variational Autoencoders)	4
1.2.1 Il Reparameterization Trick	5
1.2.2 Ottimizzazione e formulazione dell'ELBO	6
1.2.3 Compressione Percettiva e Limitazioni Qualitative	7
1.3 Flow-based models	8
1.3.1 Vincoli strutturali e limitazioni	9
1.4 Modelli di diffusione: Architettura, funzionamento e com- ponenti fondamentali	9
1.4.1 Il Processo Forward (Diffusione Diretta)	10
1.4.2 Il Processo Backward (Diffusione Inversa) e la Fun- zione di Perdita	11
1.4.3 Latent Diffusion Models (LDM)	12
1.4.4 Lo Spazio Latente e il ruolo del VAE	12
1.4.5 La rete U-Net e la stima del rumore	13
1.4.6 Condizionamento e meccanismo di Cross-Attention	15
1.4.7 L'architettura Diffusion Transformer (DiT)	16
1.5 Stato dell'arte	18

2	Struttura generale del workflow proposto	21
2.1	Evoluzione dei modelli text-to-image	22
2.1.1	Stable Diffusion 1.5	23
2.1.2	Z-Image	25
2.1.3	Flux.1	27
2.1.4	Flux.2	29
2.2	Tecniche di Fine-Tuning efficiente: Il metodo LoRA (Low-Rank Adaptation)	32
2.2.1	Fondamenti Matematici e Decomposizione a Basso Rango	32
2.2.2	Inizializzazione, Dinamica di Scala e Parametri Chiave	34
2.2.3	Ottimizzazione Computazionale e vantaggi	35
2.3	L'ecosistema ComfyUI	36
2.3.1	Componenti Fondamentali del Grafo Generativo	36
2.3.2	Il KSampler	37
3	Pipeline Sperimentale e Prototipazione Locale	41
3.1	L'Ambiente Software di Prototipazione: Kohya_ss	41
3.1.1	Gestione delle Epoche, dei Batch e delle Dinamiche Temporalì	42
3.1.2	Ottimizzazione del Learning Rate e Scheduling	43
3.1.3	Efficienza Hardware e Contenimento della VRAM	43
3.1.4	Dalla GUI alla Linea di Comando	44
3.2	Ingegnerizzazione e Creazione del Dataset	45
3.2.1	Rendering dei Soggetti	45
3.2.2	Risoluzione dei Conflitti di Retrocompatibilità dei soggetti	48
3.3	Strategie di Captioning e Analisi della Regolarizzazione	48
3.3.1	Il Fenomeno del Language Drift e l'Uso delle Class Images	49
3.4	Dinamica delle Ripetizioni ed Equazioni di Training	50
3.5	Training Preliminare in Locale: Configurazione di Kohya_ss	51

3.5.1	Configurazione dei parametri di Rete	51
3.6	Analisi Quantitativa e Qualitativa dei Risultati Locali . .	53
3.6.1	Analisi della Funzione di Perdita e delle Epoche .	53
3.6.2	Valutazione degli Output Grafici in ComfyUI . .	54
3.7	Limiti Strutturali e Analisi Combinatoria (Multi-LoRA)	56
3.7.1	Il Fenomeno della Saturazione Latente e dell'Interferenza dei Pesi	56
3.7.2	Validazione dei Moduli Singoli e Strategie di Mitigazione	57
4	L'infrastruttura HPC	59
4.1	L'Accesso a Leonardo e l'Infrastruttura di Calcolo	59
4.1.1	Nodi di Login e Politiche di Accesso	59
4.1.2	Caratteristiche Architetture dei Nodi GPU . . .	60
4.1.3	Nodi di Calcolo e Gestione delle Code via Slurm .	61
4.2	Orchestrazione Remota e Servizi Web: Chappyner	62
4.3	Evoluzione degli Strumenti di Training: da sd-scripts a musubi-tuner	63
4.3.1	I Limiti di sd-scripts e la Transizione a musubi-tuner	64
4.4	Ingegnerizzazione dell'Ambiente di Calcolo Offline	65
4.4.1	Strategie di Trasferimento Asset e Risoluzione dei Conflitti dei Tokenizer	65
4.5	Strategie di Captioning su HPC: Tagging vs Prosa	66
4.5.1	Captioning Discreto: WD14 e l'Ottimizzazione ONNX Runtime	66
4.5.2	Vision-Language Models per Architetture Transformer: Florence-2	67
5	Addestramento su Leonardo e Analisi Sperimentale	69
5.1	Ingegnerizzazione del Training	69
5.1.1	Logica di Controllo a Runtime	69
5.1.2	Allocazione Hardware Distribuita in Modalità Multi-GPU	71

5.2	Analisi e Ottimizzazione dei Training	72
5.2.1	Flux.2	72
5.2.2	Flux.1	77
5.2.3	Z-Image	79
5.2.4	Sintesi Comparativa dei Processi di Addestramento	83
5.3	Prompt Engineering e Ingegnerizzazione Sintattica in Am- bienti DiT	85
5.3.1	Strutturazione Gerarchica del Prompt e Analisi del- la Resa Materica	85
5.3.2	Test di Generalizzazione Spaziale e Iniezione di Stili Esogeni	87
5.4	Tecniche di Inpainting	88
5.4.1	Inpainting tramite Mask Editor	89
5.4.2	Inpainting tramite Workflow apposito	90
5.5	Integrazione Simultanea di Multi-LoRA	92
5.5.1	Flux.2	93
5.5.2	Flux.1	96
5.5.3	Z-Image	98
5.6	Analisi Comparativa di Sampler e Scheduler: Coerenza Visiva e Stabilità Metrica	101
5.6.1	Metodologia di Testing e Risultati Empirici . . .	101
5.6.2	Interpretazione Architetture del Comportamento dei Tensori	103
6	Analisi e utilizzo di modelli per generazione video	105
6.1	Configurazione dell'ambiente locale: Pinokio	107
6.1.1	Architettura e caratteristiche	108
6.2	Analisi strutturale dei modelli generativi video open-weights	109
6.2.1	I modelli Wan	110
6.2.2	Wan 2.1 vs Wan 2.2	111
6.2.3	LTX 2	112
6.3	Wan2GP	114
6.3.1	Caratteristiche e funzionalità	114

6.3.2	Test operativi e limiti qualitativi	115
6.4	Generazione su Leonardo: LTX 2	117
6.4.1	Impostazioni Video	117
6.4.2	Gestione dello spazio latente	119
6.4.3	Sampling	121
6.4.4	Fase di Upscaling	123
6.4.5	Movimento di Camera	124
6.4.6	Generazione dell'output	125
6.4.7	Risultati	126
6.5	Generazione su Leonardo: Wan 2.2	126
6.5.1	Workflow principale	128
6.5.2	Esplorazione dei workflow sperimentali	131
6.5.3	Gestione dell'ultimo frame	136
6.5.4	Valutazione dei workflow audio in ComfyUI	138
6.5.5	Sintesi vocale: ElevenLabs	139
6.6	Valutazione dei risultati	141
	Conclusioni	143
	Sviluppi futuri	148
	Glossario	151
	A Codice Particolarmente significativo	153
	Bibliografia e Sitografia	158

Elenco delle figure

1.1	Overview dei tipi diversi di modelli generativi	2
1.2	Architettura di una GAN	4
1.3	Architettura del VAE	7
1.4	Architettura di un Flow Based Model	9
1.5	Processo backward	11
1.6	Architettura Latent Diffusion Models	12
1.7	Architettura di una U-Net	15
1.8	Architettura DiT	17
2.1	Workflow logico generale della proposta, dal modello 3D grezzo in Blender alla sintesi dell'immagine filologica fino alla generazione dell'output video animato.	22
2.2	Schema esempio di generazione con SD 1.5	25
2.3	Architettura Z-Image	27
2.4	Architettura LoRA	33
2.5	Nodo Ksampler	39
2.6	Nodo VAE Decode	40
3.1	Esempi di render dei soggetti	47
3.2	Workflow per Stable Diffusion 1.5	55
3.3	Esempi di generazione effettuati utilizzando l'epoca 5 (sinistra) e l'epoca 10 (destra).	55
3.4	Esempio di output generato tramite configurazione ad innesto Multi-LoRA.	56

3.5	Validazione e resa geometrica del modulo LoRA singolo applicato al Tempio Etrusco.	58
4.1	Workflow di ComfyUI: Florence-2 per il captioning . . .	68
5.1	Flusso della pipeline di addestramento distribuito e inferenza su infrastruttura HPC Leonardo.	70
5.2	Comando di Addestramento Flux.2	74
5.3	Risultati di generazione per il soggetto Ati con modulo LoRA Flux.2.	77
5.4	Comando di Addestramento Flux.1	78
5.5	Risultati di generazione per il soggetto Ati con modulo LoRA Flux.1.	79
5.6	Struttura delle cartelle locali per l'addestramento offline di Z-Image.	80
5.7	Comando di Addestramento Z-Image	81
5.8	Risultati di generazione dei soggetti estratti con Z-Image Base.	83
5.9	Output di Ati ottenuto con il prompt citato sopra utilizzando Flux.2	86
5.10	Ati in un background marittimo	88
5.11	Nodo per maschera in ComfyUI	90
5.12	Inpainting con maschera manuale	90
5.13	Workflow ComfyUI - Inpainting Flux.2	91
5.14	Inpainting con il secondo metodo	92
5.15	Workflow Flux.2	94
5.16	Analisi dell'interferenza semantica nella combinazione dei moduli LoRA (Ati + Antefissa) in Flux.2.	95
5.17	Output generativi risultanti dalla combinazione sequenziale dei moduli LoRA per la Tomba Etrusca e il Tempio Etrusco in associazione al soggetto Ati (Flux.2).	96
5.18	Workflow Flux.1	96

5.19	Effetti di sovrapposizione concettuale e allucinazioni formali in scenari multi-LoRA su architettura Flux.1.	98
5.20	Workflow Z-Image.	99
5.21	Fenomeni di bleeding materico e degradazione stilistica nell'interazione multi-LoRA su Z-Image.	100
5.22	Generazione con Euler-Simple	104
5.23	Esempio di test negativo: Euler-Karras	104
6.1	Architettura complessiva della pipeline di generazione video, ramificazione strategica tra l'ambiente di testing locale (Pinokio) e la produzione su infrastruttura HPC (ComfyUI su Leonardo) con distinzione dei flussi audio.	106
6.2	Flusso Pinokio	109
6.3	Architettura LTX 2	113
6.4	workflow LTX2 - Video Settings	119
6.5	workflow LTX2 - Gestione dello spazio latente	121
6.6	workflow LTX2 - Gestione dello spazio latente	122
6.7	LTX 2 workflow - Upscaling	124
6.8	LTX 2 workflow - Output	126
6.9	Frame iniziali del video	127
6.10	Frame intermedi del video	128
6.11	WAN 2.2 - workflow principale	129
6.12	Nodo WanFirstLastFrameToVideo	132
6.13	WAN 2.2 - gestione della telecamera	133
6.14	WAN 2.2 - gestione audio	134
6.15	WAN 2.2 - Estensione video con audio	135
6.16	WAN 2.2 - Control Video	135
6.17	Live Portrait - modifica espressione tramite video	137
6.18	WAN 2.2 - Expression Editor	138
6.19	ComfyUI - Generazione audio	139
6.20	ElevenLabs - Sintesi vocale	140

Capitolo 1

Modelli di Intelligenza Artificiale Generativa

I modelli di diffusione (*Diffusion Models*) rappresentano l'attuale stato dell'arte nell'ambito della generazione di contenuti visivi. Prima di analizzare nel dettaglio il funzionamento dei modelli di diffusione, è essenziale contestualizzarne l'ascesa all'interno del panorama dell'Intelligenza Artificiale generativa. Lo sviluppo di architetture capaci di apprendere e ricostruire distribuzioni di dati complessi è stato segnato, nell'ultimo decennio, da tre paradigmi principali, quali le reti antagoniste generative (GAN), gli autoencoder variazionali (VAE) e i modelli basati su flussi normalizzanti (*Flow-based models*). Sebbene abbiano ottenuto risultati significativi, ciascuno di essi presenta limiti strutturali che hanno spinto la ricerca verso nuove direzioni.

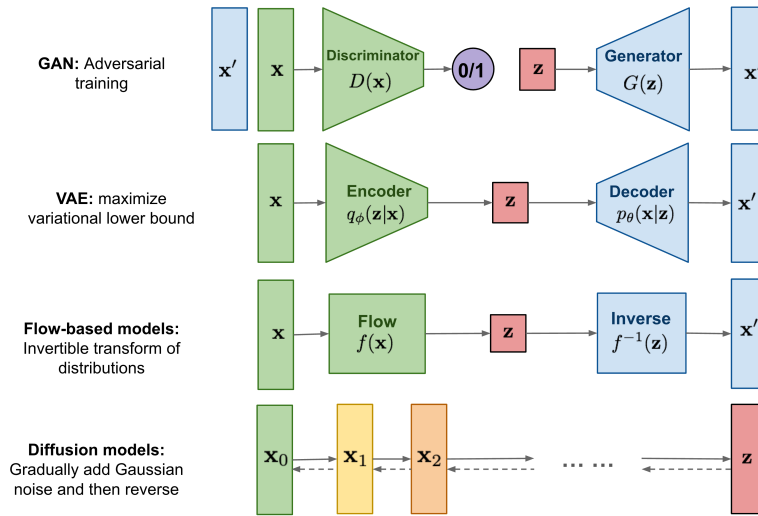


Figura 1.1: Overview dei tipi diversi di modelli generativi

1.1 Generative Adversarial Networks (GAN)

Introdotte da Ian Goodfellow et al. nel 2014, le Reti Antagoniste Generative (*Generative Adversarial Networks*) [21] hanno rappresentato per anni il paradigma dominante nella sintesi di immagini ad alta fedeltà. A differenza dei modelli che tentano di calcolare esplicitamente la densità di probabilità dei dati, le GAN adottano un approccio implicitamente basato sulla teoria dei giochi, formalizzando l'addestramento utilizzando due modelli:

- **Il Generatore (G):** riceve in *input* un vettore di rumore casuale z estratto da una distribuzione nota e tenta di mapparlo nello spazio dei dati reali, producendo un campione sintetico $G(z)$. L'obiettivo del Generatore è massimizzare la probabilità che il Discriminatore commetta un errore di valutazione.
- **Il Discriminatore (D):** agisce come un classificatore binario. Riceve in *input* un campione x e restituisce un valore scalare $D(x)$ in $[0, 1]$ che rappresenta la probabilità stimata che il campione esaminato sia autentico e non artificiale.

Questi due modelli competono tra loro durante il processo di addestramento, il generatore cerca di ingannare il discriminatore, mentre il modello critico cerca di non farsi ingannare. Questo interessante gioco a somma zero tra i due modelli li motiva entrambi a migliorare le proprie funzionalità.

Matematicamente, questo impianto architetturale viene ottimizzato attraverso una funzione di minimax (*minimax objective function*), formalizzata dall'equazione:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (1.1)$$

Durante l'addestramento, il Discriminatore tenta di massimizzare la funzione per diventare sempre più accurato nel riconoscere i falsi, mentre il Generatore si focalizza sul minimizzare la componente logaritmica, affinando la propria capacità di generare dettagli visivi realistici. In una condizione ideale di convergenza teorica, l'ottimizzazione raggiunge l'Equilibrio di Nash, punto in cui il Generatore produce immagini indistinguibili dal *dataset* originale e il Discriminatore si stabilizza su una probabilità di predizione puramente casuale pari a 0.5.

1.1.1 Limitazioni intrinseche e instabilità matematica

Nella pratica, l'ottimizzazione di una funzione minimax presenta severe barriere matematiche. Quando il Discriminatore si comporta in modo troppo efficiente nell'addestramento, il gradiente dell'errore trasmesso al Generatore si azzerava (*vanishing gradient*), interrompendo bruscamente il processo di apprendimento.

Il fallimento più comune delle GAN è il *Mode Collapse*. Invece di apprendere l'intera varietà della distribuzione di *input*, il Generatore scopre un ristretto sottoinsieme di configurazioni (*mode*) che riesce a ingannare sistematicamente il Discriminatore. Di conseguenza, la rete inizia a

produrre ripetutamente campioni quasi identici, perdendo totalmente la diversità del *dataset*.

Per evitare che una rete prevalga sull'altra, l'addestramento richiede un bilanciamento dei tassi di apprendimento (*learning rates*) delle architetture e delle penalità di regolarizzazione. Questa estrema volatilità rende le GAN poco scalabili e difficilmente applicabili a *pipeline* industriali stabili o a infrastrutture HPC su larga scala senza un monitoraggio costante.

Questi limiti strutturali e la fragilità intrinseca della funzione di perdita minimax hanno spinto la comunità scientifica a esplorare funzioni basate su metriche geometriche differenti, come la *Wasserstein Distance* nelle WGAN, fino ad arrivare al superamento definitivo del concetto avversariale tramite l'introduzione dei processi iterativi dei modelli di diffusione.

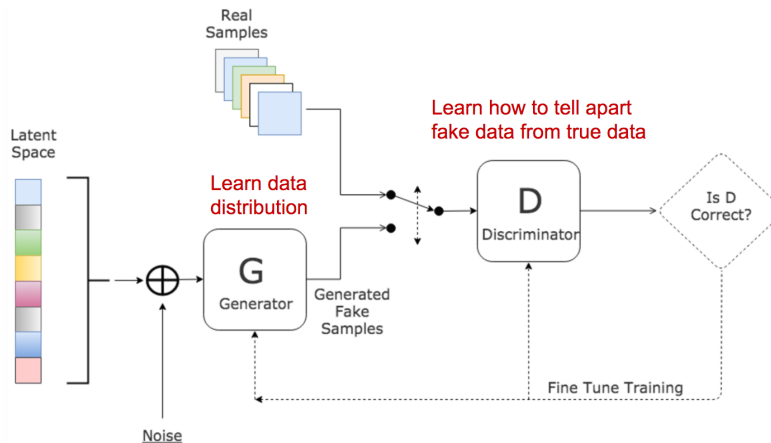


Figura 1.2: Architettura di una GAN

1.2 VAE (Variational Autoencoders)

I *Variational Autoencoders* (VAE) [50, 25, 26], introdotti da Kingma e Welling nel 2013, rappresentano un'estensione in chiave probabilistica del classico paradigma degli autoencoder deterministici. Sono modelli di *deep learning* composti da un *encoder* che impara a isolare le variabili

latenti più importanti dai dati di addestramento e da un *decoder* che poi utilizza quelle variabili latenti per ricostruire i dati di *input*.

Un autoencoder tradizionale mappa un'immagine di *input* in un singolo punto fisso ed isolato dello spazio latente, rendendo l'architettura idonea alla sola compressione e ricostruzione deterministica dei dati. Un VAE modella le variabili latenti come una distribuzione di probabilità continua che introduce la capacità di campionare nuove coordinate all'interno dello spazio latente, elevando l'architettura a vero e proprio modello generativo.

L'architettura si articola in due macro-componenti speculari, generalmente implementati tramite reti neurali convoluzionali (CNN) per sfruttare le capacità intrinseche di estrazione gerarchica delle caratteristiche visive:

- **Encoder:** riceve l'*input* x e approssima la distribuzione posteriore delle variabili latenti z . Nella configurazione standard tale distribuzione viene assunta come una gaussiana multivariata, motivo per cui l'*encoder* non restituisce un vettore statico, bensì i parametri matematici della distribuzione, ovvero il vettore della media e della varianza.
- **Decoder:** riceve in *input* un vettore latente z campionato dalla distribuzione calcolata dall'*encoder* e tenta di ricostruire l'*input* originario sotto forma di *output* sintetico $\hat{x}$.

1.2.1 Il Reparameterization Trick

Una proprietà intrinseca della casualità è che non può essere ottimizzata. Un vettore di valori casuali non ha una derivata, quindi nessun gradiente che esprima alcun *pattern* negli *output* del modello risultante e non può essere ottimizzato tramite la *backpropagation* utilizzando una forma di discesa del gradiente. Ciò significherebbe che una rete neurale che utilizza il processo di campionamento casuale non può apprendere i parametri ottimali per svolgere il proprio compito.

Per aggirare questo ostacolo, i VAE utilizzano il trucco della riparametrizzazione. Si introduce un parametro (ϵ) che è un valore casuale selezionato dalla distribuzione normale tra 0 e 1. Così si riparametrizza la variabile latente z come:

$$z = \mu + \sigma \odot \epsilon \quad (1.2)$$

Viene scelto un valore per la variabile latente z iniziando con la media di quella variabile (rappresentata da μ) e spostandola di un multiplo casuale (rappresentato da ϵ) di una deviazione standard (σ). Condizionato da quel valore specifico di z , il *decoder* emette un nuovo campione.

1.2.2 Ottimizzazione e formulazione dell'ELBO

L'addestramento di un VAE si basa sulla massimizzazione dell' *Evidence Lower Bound* (ELBO), un limite inferiore della verosimiglianza dei dati che funge da funzione di perdita multi-obiettivo.

La *loss* complessiva della rete si scompone in due termini matematici distinti e in competizione tra loro:

- **Reconstruction Loss** ($-E_q$): misura la discrepanza tra l'*output* generato $\hat{x}$ e l'immagine originale x , solitamente implementata tramite l'errore quadratico medio (MSE). Questo termine spinge il modello a preservare la massima quantità di informazioni possibile per garantire ricostruzioni fedeli.
- **Kullback-Leibler Divergence** (D_{KL}): agisce come termine di regolarizzazione. Essa calcola la distanza statistica tra la distribuzione latente stimata dall'*encoder* $q_\phi(z|x)$ e una distribuzione a priori standard $p(z)$, assunta per convenzione come una gaussiana standard.

La divergenza di KL penalizza la rete se l'*encoder* tenta di raggruppare le codifiche in regioni disgiunte e isolate, forzando lo spazio latente

a configurarsi come continuo, compatto e privo di lacune. In questo modo, punti vicini nello spazio latente riflettono caratteristiche semantiche e geometriche simili nello spazio dei pixel. Questo garantisce interpolazioni fluide tra i soggetti generati e fornisce un *manifold* strutturato su cui far operare i modelli di diffusione.

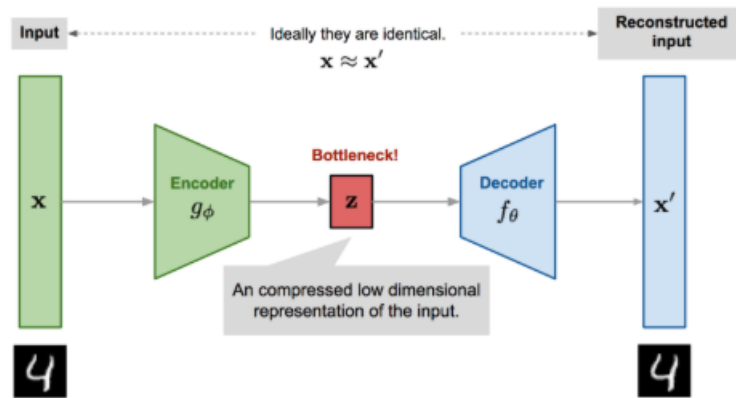


Figura 1.3: Architettura del VAE

1.2.3 Compressione Percettiva e Limitazioni Qualitative

Nel contesto delle moderne *pipeline* generative, il VAE non viene utilizzato per produrre direttamente l'immagine finale, ma funge da fondamentale strumento di compressione percettiva. Il suo compito primario consiste nel filtrare il rumore ad alta frequenza e le informazioni ridondanti, preservando esclusivamente i vettori semantici essenziali del soggetto. Questo processo riduce drasticamente la dimensionalità spaziale dei dati, abbattendo in modo significativo il carico computazionale.

Tuttavia, un limite strutturale ben noto dei VAE risiede nella produzione di ricostruzioni intrinsecamente sfocate o comunque strettamente legate alla qualità degli *input*. Questa problematica deriva dalla natura stessa della *reconstruction loss* standard basata su pixel (MSE) che penalizza pesantemente le grandi deviazioni, ma tende a fare la me-

dia matematica di tutte le possibili configurazioni dei dettagli ad alta frequenza.

Si crea così un compromesso: mentre la regolarizzazione tramite divergenza di KL garantisce uno spazio latente continuo, la perdita statistica di nitidezza richiede l'integrazione di componenti aggiuntive nelle *pipeline* complesse per recuperare il livello di definizione e fotorealismo necessario alle applicazioni ad alta fedeltà.

1.3 Flow-based models

I modelli generativi basati su flusso (*Flow-based Generative Models*) [16, 49], formalizzati da Rezende e Mohamed nel 2015, rappresentano il terzo paradigma classico della generazione di contenuti multimediali. Sfruttano una sequenza di funzioni invertibili e derivabili, chiamate flusso normalizzante (*Normalizing Flow*), per mappare una distribuzione di dati complessa x in una distribuzione latente semplice e nota z (solitamente una gaussiana standard).

Il nucleo matematico di questo approccio risiede nel teorema del cambio di variabile. Se la trasformazione tra lo spazio dei dati e lo spazio latente è biiettiva ed espressa da una funzione $z = f(x)$, la densità di probabilità esatta di un campione x può essere calcolata attraverso l'equazione:

$$p_X(x) = p_Z(f(x)) \cdot \left| \det \frac{\partial f(x)}{\partial x} \right| \quad (1.3)$$

Dove la derivata rappresenta la matrice Jacobiana della trasformazione. Durante la fase d'addestramento, il modello ottimizza direttamente la log-verosimiglianza esatta dei dati, garantendo una convergenza stabile e priva di fenomeni di *mode collapse*. In fase di generazione (campionamento), il processo viene invertito. Si estrae un vettore casuale z dalla gaussiana latente e lo si fa passare attraverso le funzioni inverse per ottenere l'immagine finale.

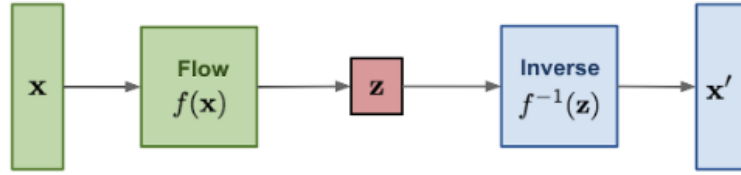


Figura 1.4: Architettura di un Flow Based Model

1.3.1 Vincoli strutturali e limitazioni

Nonostante la solidità matematica del calcolo esatto della verosimiglianza, i modelli basati su flusso presentano forti limitazioni pratiche che ne hanno impedito la scalabilità su larga scala. Infatti, la trasformazione $f(x)$ deve essere rigorosamente invertibile e il determinante della sua matrice Jacobiana deve essere facile e veloce da calcolare. Questo impone vincoli geometrici severissimi agli strati della rete neurale, limitandone la flessibilità espressiva.

Inoltre, poiché la trasformazione deve preservare la dimensionalità, i modelli basati su flusso tendono a essere estremamente pesanti in termini di parametri e memoria VRAM, risultando inefficienti per la generazione di immagini ad alta risoluzione.

1.4 Modelli di diffusione: Architettura, funzionamento e componenti fondamentali

Nel panorama dell'Intelligenza Artificiale generativa, i Modelli di Diffusione (*Diffusion Models*) rappresentano l'attuale stato dell'arte per la sintesi di contenuti multimediali complessi. Essi si sono affermati come la risposta evolutiva e il superamento delle criticità biologiche dei modelli classici. A differenza delle GAN, i modelli di diffusione beneficiano di una procedura di addestramento fissa, stabile e priva di dinamiche avversariali distruttive. Rispetto ai VAE e ai modelli a flusso, essi non impongono vincoli geometrici restrittivi o trasformazioni biettive rigide,

operando su architetture neurali altamente flessibili capaci di catturare dettagli visivi e fedeltà semantica precedentemente inaccessibili.

L'impianto teorico introdotto originariamente da Sohl-Dickstein et al. nel 2015 e perfezionato da Ho et al. nel 2020 con i *Denoising Diffusion Probabilistic Models* (DDPM), trae profonda ispirazione dai principi della termodinamica statistica di non equilibrio. Il sistema simula il processo stocastico attraverso cui le particelle si diffondono all'interno di un fluido aumentando l'entropia, fino a distruggere qualsiasi struttura macroscopica. Nel contesto della *computer vision*, questo principio viene applicato trasformando progressivamente un'immagine reale in rumore gaussiano puro, per poi addestrare una rete neurale a compiere l'esatto cammino inverso, ovvero rimuovere il rumore in modo iterativo e controllato per far emergere un'immagine coerente dal caos numerico.

Il meccanismo si articola in due fasi principali, definite come processi di catene di Markov: processo *Forward* e *Backward*.

1.4.1 Il Processo Forward (Diffusione Diretta)

Nella fase *forward*, il modello prende un'immagine reale dal *dataset* di addestramento e vi aggiunge progressivamente piccole quantità di rumore gaussiano lungo una sequenza di centinaia o migliaia di passi temporali discreti (*timesteps*). Questo degrado segue una pianificazione programmata della varianza (*variance schedule*), che stabilisce la quantità esatta di disturbo da inserire in ogni singolo *step*.

All'inizio del processo, l'immagine è perfettamente nitida e andando avanti lungo i passi temporali il segnale visivo originale viene sistematicamente eroso. Al termine della catena, l'informazione strutturale e i legami semantici di partenza sono completamente distrutti, lasciando spazio unicamente a un blocco di rumore bianco puro. Questa fase non richiede una rete neurale attiva per essere calcolata, poiché si basa su un'operazione statistica predefinita.

1.4.2 Il Processo Backward (Diffusione Inversa) e la Funzione di Perdita

La fase *backward* costituisce il vero nucleo generativo del sistema ed è il momento in cui entra in gioco l'addestramento della rete neurale. Poiché non è possibile invertire matematicamente il rumore in modo diretto, la rete viene addestrata a compiere una stima, per ogni singolo passo temporale, analizza l'*input* corrotto e tenta di indovinare la quantità esatta di rumore che è stata aggiunta nel rispettivo passaggio *forward*.

Durante l'ottimizzazione, la funzione di perdita (*loss function*) calcola lo scarto tra il rumore reale introdotto nella fase diretta e il rumore predetto dalla rete. Minore è questo scarto, più il modello diventa efficiente nel ripulire i dati. Una volta completato l'addestramento, il modello possiede la capacità di generare contenuti completamente nuovi. Partendo da un *input* composto esclusivamente da rumore casuale generato dal computer, la rete applica iterativamente il proprio algoritmo di rimozione (*denoising*). Passo dopo passo, il disturbo viene sottratto e le caratteristiche strutturali dell'immagine iniziano a emergere, fino a materializzare un'immagine finale ad alta fedeltà. Nelle *pipeline* moderne, questa traiettoria inversa non è casuale, ma viene guidata da segnali esterni (come *prompt* testuali) che orientano la ricostruzione verso il soggetto desiderato.

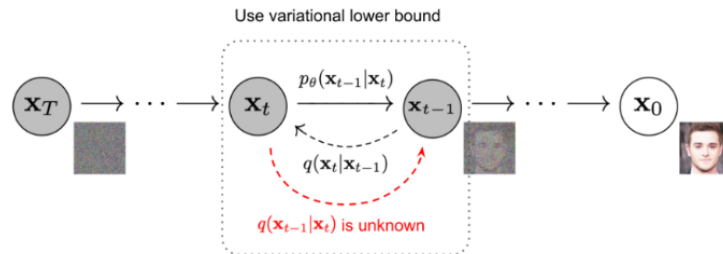


Figura 1.5: Processo backward

1.4.3 Latent Diffusion Models (LDM)

Operare direttamente sullo spazio nativo dei pixel dell'immagine richiederebbe risorse computazionali proibitive e una quantità di VRAM insostenibile per immagini ad alta risoluzione. Per risolvere questo problema, i moderni sistemi (come Stable Diffusion o Flux) adottano il paradigma dei *Latent Diffusion Models* (LDM) [36], formalizzato da Rombach et al. nel 2022.

In questa architettura, i processi di diffusione e *denoising* descritti in precedenza vengono interamente traslati all'interno di uno spazio geometrico compresso a bassa dimensionalità. Il sistema smette di operare sulla griglia visibile dei pixel e si fonda su tre macro-componenti interconnesse che ottimizzano l'efficienza senza intaccare la qualità semantica dell'*output*. Questi componenti vengono descritti nei paragrafi successivi.

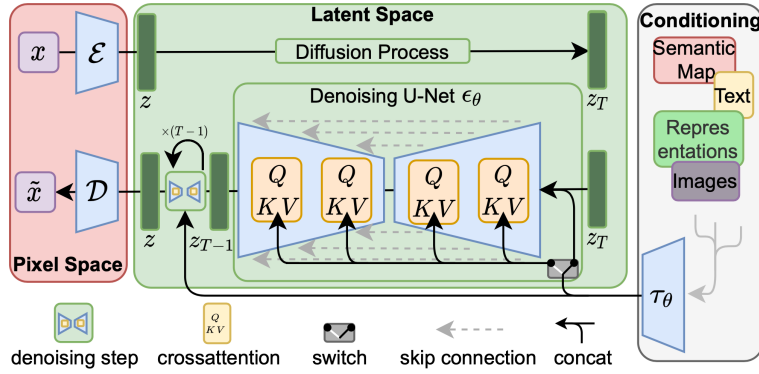


Figura 1.6: Architettura Latent Diffusion Models

1.4.4 Lo Spazio Latente e il ruolo del VAE

Come analizzato nel paragrafo 1.2, il VAE nasce storicamente come modello generativo indipendente. Nei sistemi di diffusione latente, tuttavia, esso viene utilizzato esclusivamente come modulo di compressione percettiva e di interfaccia geometrica.

- **L'Encoder (E):** riceve l'immagine ad alta risoluzione x nello spazio dei pixel e la comprime in una mappa latente $z = E(x)$, ri-

ducendo la dimensionalità spaziale ma espandendo la profondità dei canali. L'encoder memorizza le caratteristiche semantiche, geometriche e topologiche essenziali del soggetto, scartando i dettagli ridondanti e ad alta frequenza per alleggerire il calcolo.

- **Il Decoder (D):** interviene unicamente al termine della *pipeline*. Prende la mappa latente finale completamente de-rumorizzata z_0 e la decompone attraverso strati di convoluzione e *upsampling*, ripristinando la risoluzione nativa in pixel e iniettando nuovamente i micro-dettagli visivi nitidi.

1.4.5 La rete U-Net e la stima del rumore

Mentre il VAE definisce l'ambiente geometrico di lavoro, la rete U-Net costituisce il vero motore algoritmico deputato alla rimozione del rumore all'interno dello spazio latente. Ideata originariamente da Ronneberger et al. nel 2015, questa architettura è stata integrata nei modelli di diffusione grazie alla sua straordinaria capacità di elaborare e ricostruire l'informazione spaziale a diversi livelli di granularità.

L'architettura si distingue per una struttura perfettamente simmetrica a forma di "U", che si sviluppa lungo due rami principali collegati da un ponte centrale:

- **Il ramo di contrazione (Encoder / Downsampling):** situato sul lato sinistro della "U", questo percorso ha l'obiettivo di comprendere cosa è presente nell'immagine. L'*input* passa attraverso l'applicazione ripetuta di blocchi convoluzionali seguiti da operazioni di *Max Pooling*. Ad ogni passaggio, la risoluzione spaziale della matrice viene dimezzata, mentre la profondità dei canali aumenta. Questo campionamento progressivo costringe la rete a scartare i dettagli microscopici locali per estrarre caratteristiche semantiche globali e di contesto a lungo raggio.
- **Il collo di bottiglia (Bottleneck):** rappresenta il punto più basso della struttura, dove l'informazione raggiunge la massima com-

pressione spaziale e la massima densità semantica. Nelle applicazioni di diffusione, è proprio in questo nucleo profondo che vengono iniettati gli *embedding* testuali per condizionare il modello.

- **Il ramo di espansione (Decoder / Upsampling):** questo percorso risale la "U" per determinare dove posizionare gli elementi, ricostruendo la mappa latente originaria. Il processo avviene tramite operazioni di deconvoluzione o *Up-sampling* che raddoppiano la risoluzione spaziale a ogni livello, riducendo parallelamente la profondità dei canali.

Il vero punto di forza di questa architettura risiede nelle *skip connections* (connessioni dirette). Durante la discesa nel ramo di contrazione, molti dettagli geometrici fini e informazioni topologiche ad alta frequenza vanno inevitabilmente perduti. Per evitare che l'immagine finale risulti sfocata o priva di coerenza strutturale, le *skip connections* prendono le mappe di caratteristiche (*feature maps*) preservate nei vari livelli dell'encoder e le copiano direttamente, concatenandole ai rispettivi blocchi speculari del decoder in fase di risalita.

Questo consente al motore di *denoising* di combinare la comprensione semantica profonda (proveniente dal basso della rete) con la precisione geometrica originaria (proveniente dall'inizio della rete), garantendo che la stima del rumore sia spazialmente accurata e filologicamente coerente con i vincoli del soggetto.

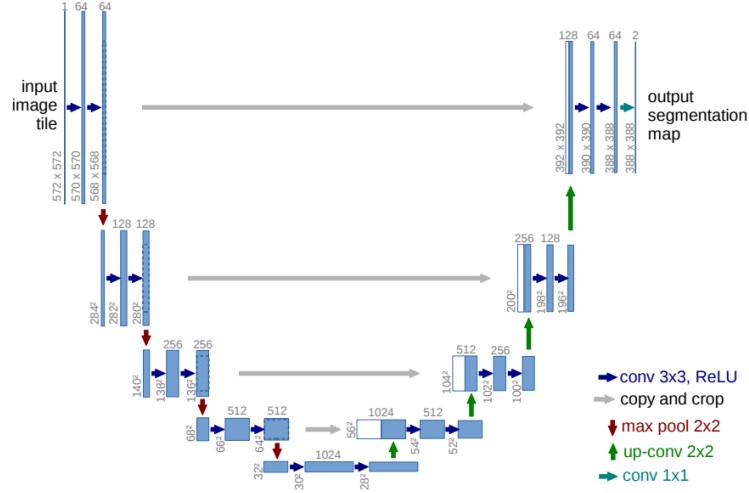


Figura 1.7: Architettura di una U-Net

1.4.6 Condizionamento e meccanismo di Cross-Attention

Nei modelli di generazione controllata guidati da *prompt* testuali (*conditional diffusion*), la U-Net deve essere direzionata affinché il processo di *denoising* non crei un'immagine casuale, ma rispetti la richiesta dell'utente.

Il testo inserito viene elaborato da un *text encoder* esterno basato su architettura *Transformer* che traduce la stringa alfanumerica in una sequenza di vettori di *embedding* semantici. All'interno dei blocchi della U-Net, i vettori latenti dell'immagine interagiscono matematicamente con gli *embedding* del testo tramite l'operazione di *Cross-Attention*:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (1.4)$$

Questo meccanismo mappa dinamicamente i concetti linguistici estratti dal *prompt* sulle corrispondenti coordinate spaziali e geometriche dell'immagine in fase di ricostruzione. È grazie a questa interazione che il modello riesce a far convergere il processo inverso verso il soggetto richiesto con estrema accuratezza.

È importante sottolineare che, sebbene l'architettura convoluzionale U-Net abbia costituito lo standard industriale per i modelli di diffusione latente, la ricerca più recente ha iniziato a convergere verso l'utilizzo di strutture basate interamente su *Transformer* (*Diffusion Transformers* o DiT).

1.4.7 L'architettura Diffusion Transformer (DiT)

Sebbene l'architettura U-Net convoluzionale abbia rappresentato lo standard di riferimento per la prima generazione di modelli di diffusione latente, l'evoluzione della letteratura scientifica ha evidenziato severi colli di bottiglia legati alla sua natura intrinseca. Le reti convoluzionali eccellono nell'elaborazione di caratteristiche locali, ma faticano a mappare relazioni geometriche globali a lungo raggio e mostrano una scarsa scalabilità (*scaling laws*) all'aumentare dei dati di addestramento e della potenza computazionale.

Per superarne tali limitazioni, uno studio condotto da Peebles e Xie (2022) ha introdotto l'architettura *Diffusion Transformer* (DiT), determinando un cambio di paradigma radicale nel campo della sintesi d'immagine. Questo approccio prevede la totale sostituzione della U-Net convoluzionale con un'infrastruttura interamente basata su *Transformer*, mutuata direttamente dai *Large Language Models* (LLM) ma adattata alla stima del rumore.

L'architettura DiT si articola su tre passaggi geometrici fondamentali:

- **Linearizzazione in Patch (Patchification):** poiché i *Transformer* elaborano dati sequenziali, l'*input* latente bidimensionale ($64 \times 64 \times 4$) non viene analizzato tramite filtri convoluzionali scorrevoli. La matrice viene invece sezionata in un reticolo di piccoli tasselli quadrati disgiunti, chiamati *patches* (tipicamente di dimensione 2×2). Ogni *patch* viene appiattita (*flattened*) in un vettore lineare e proiettata in uno spazio ad alta dimensionalità, venendo trattata a tutti gli effetti come se fosse un *visual token*.

- **Embedding Posizionale:** a differenza delle convoluzioni, che mantengono l'ordine dei pixel grazie alla loro griglia rigida, i *Transformer* sono del tutto privi di informazioni innate sulla posizione dei *token*. Per preservare la topologia dell'immagine, ai vettori delle *patch* viene sommato un *Positional Embedding* bidimensionale, che assegna a ogni tassello una coordinata spaziale matematica univocamente determinata.
- **Blocchi di Self-Attention Consistenti:** i *token* visivi così strutturati attraversano una sequenza di blocchi *Transformer* standard. Grazie al meccanismo di *Multi-Head Self-Attention*, ogni singola *patch* dell'immagine è in grado di calcolare la propria relazione rispetto a tutte le altre *patch* contemporaneamente, indipendentemente dalla loro distanza geometrica.

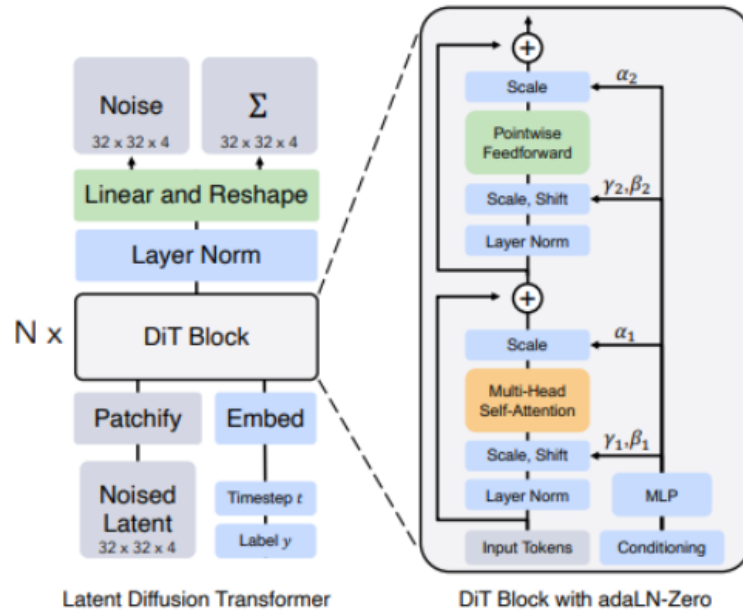


Figura 1.8: Architettura DiT

L'introduzione dei DiT comporta benefici di enorme rilievo per gli scopi scientifici e filologici di questa tesi:

- **Conservazione delle rigidzze Geometriche:** la capacità dell’auto-attenzione di calcolare relazioni spaziali globali permette al modello di preservare la perfetta simmetria di un manufatto archeologico o le rigide proporzioni prospettiche di un’architettura storica, laddove le convoluzioni locali della U-Net tendono a produrre distorsioni o allucinazioni fluide.
- **Scalabilità prevedibile:** i DiT rispondono in modo eccellente alle leggi di scala. All’aumentare del numero di parametri del modello, la qualità visiva, l’aderenza al *prompt* testuale e la precisione del dettaglio microscopico crescono in modo lineare e costante.

Questo impianto teorico costituisce la base tecnologica per i modelli di frontiera esaminati nei capitoli successivi, consentendo di comprendere l’evoluzione strutturale da Z-Image fino alla complessità parametrica della famiglia Flux.

1.5 Stato dell’arte

I modelli di Intelligenza Artificiale generativa analizzati nei paragrafi precedenti stanno trovando una rapida e capillare diffusione all’interno del settore archeologico, della museologia e della salvaguardia del patrimonio culturale.

Allo stato attuale, piattaforme *open-source* di condivisione come CivitAI ospitano numerosi modelli e pesi aggiuntivi addestrati su stili artistici o archeologici generali. Tuttavia, la loro natura commerciale produce artefatti visivi affetti da anacronismi storici, rendendoli inadatti per scopi scientifici o divulgativi. Per ovviare a tale limite, la letteratura scientifica recente si è focalizzata sullo sviluppo di *framework* personalizzati basati sull’estensione LoRA (*Low-Rank Adaptation*). Questa tecnica permette di iniettare competenze stilistiche, geometriche e filologiche all’interno di modelli di diffusione pre-addestrati senza doverne riaddestrare l’intero corpo di parametri, ottimizzando l’accuratezza su *dataset* ristretti e specifici.

Altri studi hanno utilizzato tecniche simili a quelle esplorate in questa tesi, applicate in modo diverso o a soggetti differenti.

L'esempio metodologico più affine agli obiettivi della presente tesi è rappresentato dallo studio “AI-Assisted Restoration of Yangshao Painted Pottery Using LoRA and Stable Diffusion”, implementato all'interno dell'ambiente di programmazione visuale ComfyUI. La ricerca si concentra sul restauro digitale della ceramica dipinta della Cultura Yangshao (circa 5000-3000 a.C.), un patrimonio caratterizzato da motivi geometrici simmetrici estremamente complessi e specifici.

Per preservare tale identità visiva, i ricercatori hanno addestrato un modulo LoRA dedicato su un dataset controllato di *pattern* integrati e storicamente verificati. Lo studio dimostra come l'introduzione del LoRA specifico all'interno della *pipeline* di *denoising* permetta al modello di ricostruire le porzioni mancanti o danneggiate dei reperti con una precisione filologica inaccessibile ai modelli di diffusione standard, i quali tenderebbero a colmare i vuoti con *pattern* inventati o incoerenti.

Nel progetto “AI-Based Pre-Renewal Design for Historic Building Facades: An AIGC-LoRA Framework” viene affrontato il problema del falso storico generato dalle IA commerciali. Questo studio evidenzia come i modelli generativi puri, se guidati unicamente da *prompt* testuali generici, tendano a produrre facciate di edifici storici visivamente coerenti ma architettonicamente errate dal punto di vista dell'epoca di riferimento.

Addestrando molteplici LoRA specialistici suddivisi per categorie architettoniche, addestrati esclusivamente su immagini selezionate da storici dell'architettura e grazie ad una buona ingegnerizzazione del *prompt*, il progetto conferma che la modularità dei LoRA è l'unico strumento in grado di vincolare la flessibilità generativa dell'IA al rispetto delle regole strutturali del restauro architettonico.

Una declinazione speculare, focalizzata sulla ricostruzione tridimensionale, è offerta dallo studio “One-to-many Reconstruction of 3D Geometry of Cultural Artifacts using a Synthetically Trained Generative Model”. A differenza degli approcci orientati al ripristino delle *texture* o delle finiture superficiali tramite diffusione e LoRA, questo *framework*

esplora la possibilità di ricostruire l'intera geometria 3D di un artefatto danneggiato partendo da un singolo *input* bidimensionale (2D). Il modello generativo viene addestrato su un *dataset* di dati sintetici (modelli 3D) per apprendere a codificare le regolarità geometriche e strutturali intrinseche di una specifica classe di manufatti. Questo studio evidenzia la versatilità degli strumenti generativi nel settore archeologico, dimostrando come essi possano operare non solo per la sintesi visiva bidimensionale, ma anche per il recupero spaziale dei volumi e delle forme originarie.

Infine, il *paper* “Infusing AI Art with Cultural Authenticity Through the Culture-Specific LoRA” evidenzia il lavoro di *training* del LoRA, vengono usati algoritmi per ripulire le immagini di *training*, isolando i veri elementi culturali ed eliminando i rumori di fondo o gli elementi non pertinenti grazie all'utilizzo di *tag* delle immagini molto ricchi e precisi. Usando questi *tag* semantici nel *prompt* si ottengono immagini di base fotorealistiche e accurate.

Capitolo 2

Struttura generale del workflow proposto

Quanto detto fino ad ora evidenzia in modo inequivocabile come l'accoppiamento tra modelli di diffusione ed estensioni LoRA personalizzate rappresenti la metodologia d'elezione per ottenere *output* ad alta fedeltà filologica.

Il lavoro svolto in questa tesi si inserisce in questo filone di ricerca, utilizzando i modelli di diffusione insieme, modelli LoRA addestrati specificatamente sui soggetti scelti e un buon *prompt engineering*. L'obiettivo si focalizza sulla realizzazione di un ecosistema integrato e scalabile basato su soluzioni interamente *open-source*, volto alla generazione coordinata non solo di utensili e architetture storiche rigorosamente attendibili a partire da database derivati da modelli 3D, ma anche di avatar e personaggi digitali dinamicamente inseriti nel medesimo contesto storico, gettando le basi per un workflow di generazione di prodotti filologicamente corretti. Questo approccio risponde alla necessità di ottenere generazioni di ottima qualità (utilizzando modelli di diffusione di nuova generazione) e di elevata fedeltà filologica (utilizzando moduli LoRA addestrati su dataset specifici).

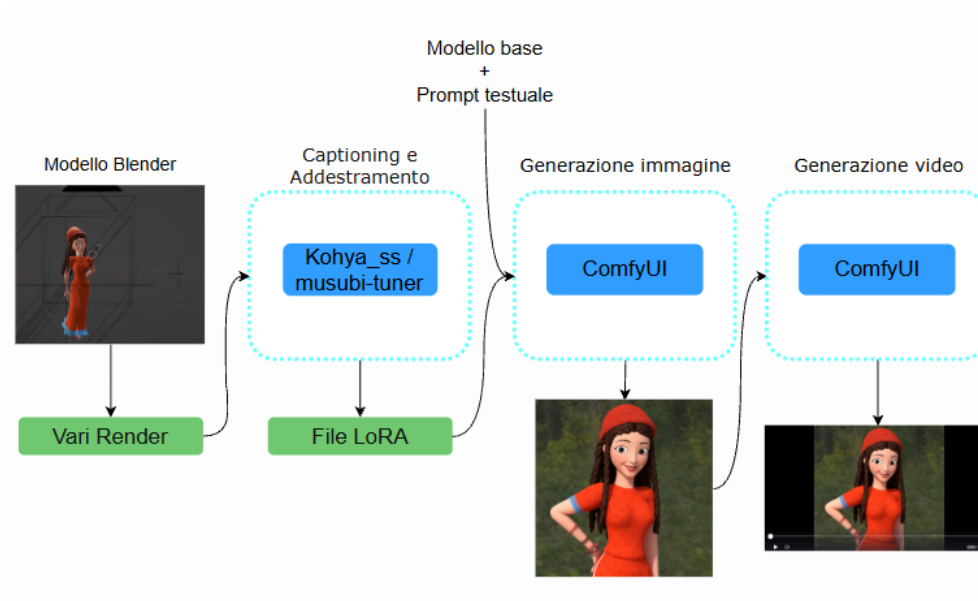


Figura 2.1: Workflow logico generale della proposta, dal modello 3D grezzo in Blender alla sintesi dell'immagine filologica fino alla generazione dell'output video animato.

Il flusso operativo (Figura 2.1) mappa in modo lineare la transizione sequenziale del dato: il rendering bidimensionale dei soggetti modellati in **Blender** funge da input per le fasi di *captioning* e addestramento dei pesi adattivi tramite i framework **Kohya_ss** e **musubi-tuner**. Successivamente, i file LoRA risultanti vengono integrati nell'ecosistema a nodi **ComfyUI** per condizionare l'inferenza dei modelli base (*text-to-image*), ottenendo l'immagine sintetica filologicamente corretta. Quest'ultima costituisce infine il punto di partenza per l'ultimo blocco di generazione e animazione video (*image-to-video*), orchestrato anch'esso all'interno dell'ambiente ComfyUI.

2.1 Evoluzione dei modelli text-to-image

I modelli utilizzati nel corso di questo lavoro sono stati Stable Diffusion 1.5 per la parte sperimentale in locale, mentre per sfruttare le potenzialità della infrastruttura HPC si sono utilizzati Z-Image, Flux.1 e Flux.2. In

questo capitolo si analizzano le strutture architetture di questi modelli, ponendo le basi per comprendere le ragioni per cui alcune di queste configurazioni garantiscono risultati qualitativamente e filologicamente superiori rispetto alle altre.

2.1.1 Stable Diffusion 1.5

Rilasciato nell'ottobre 2022 da RunwayML (in collaborazione con Stability AI e CompVis), Stable Diffusion 1.5 (SD 1.5) [35] rappresenta il pilastro storico e il punto di partenza della sperimentazione di questa tesi. Pur essendo un'architettura superata dalle successive generazioni di modelli, essa rimane comunque un riferimento per i test locali su *hardware consumer* grazie alla sua eccezionale leggerezza computazionale e al vastissimo ecosistema di estensioni della *community*.

Dal punto di vista della struttura, SD 1.5 è un modello di diffusione latente composto da tre blocchi principali interconnessi, configurati con pesi e parametri ben precisi:

- **Il Text Encoder:** per elaborare e comprendere le descrizioni testuali inserite dall'utente, il modello utilizza un codificatore pre-addestrato da OpenAI, nello specifico CLIP ViT-L/14 congelato (*frozen*). Questo modulo ha una dimensione fissa di 123 milioni di parametri e si sviluppa su una geometria interna caratterizzata da 12 strati (*layers*), un'ampiezza del vettore di *embedding* (*width*) di 768 unità e 12 teste di attenzione (*heads*). Presenta un vincolo strutturale rigido, accetta un limite massimo di 77 *token* in *input*. Questo *encoder* traduce il *prompt* in un vettore di *embedding* semantico che guiderà la generazione.
- **Il Denoiser (U-Net):** il nucleo computazionale deputato alla stima del rumore è una rete U-Net condizionata, caratterizzata da una dimensione di circa 860 milioni di parametri. Essendo un'architettura basata su operazioni convoluzionali, essa elabora le informazioni focalizzandosi sulle relazioni spaziali locali della matrice dell'im-

magine. La stima avviene tramite meccanismi di *Cross-Attention* che incrociano ad ogni blocco i vettori spaziali dell'immagine con i vettori semantici del testo estratti da CLIP.

- **L'Autoencoder (VAE):** Per ridurre l'impatto sulla memoria VRAM, l'intero processo di diffusione avviene all'interno dello spazio latente del VAE, che comprime le immagini applicando un fattore di sottocampionamento pari a 8, per cui un'immagine nativa da 512×512 pixel viene elaborata come una matrice latente da $64 \times 64 \times 4$.

In totale, l'architettura complessiva di SD 1.5 si attesta appena sotto il miliardo di parametri, circa 983 milioni. Se da un lato questa compattezza le permette di eseguire l'inferenza e l'addestramento di moduli LoRA personalizzati in locale richiedendo meno di 8 GB di VRAM, dall'altro introduce forti limitazioni per gli obiettivi di questa tesi:

- **Risoluzione nativa ridotta:** il modello è stato addestrato sul *dataset* LAION-5B per generare nativamente immagini a una risoluzione di soli 512×512 pixel. Tentare di generare formati più grandi causa sistematicamente artefatti, duplicazioni di soggetti e distorsioni geometriche.
- **Debolezza semantica:** l'encoder CLIP da 123M di parametri possiede una comprensione linguistica limitata. Non è in grado di comprendere relazioni spaziali complesse e soffre fortemente quando si inseriscono termini tecnici, archeologici o d'epoca specifici, tendendo a ignorarli o a interpretarli in chiave moderna e commerciale.

Questo set di limiti giustifica la necessità di scalare la sperimentazione verso l'infrastruttura HPC per impiegare modelli di scala nettamente superiore.

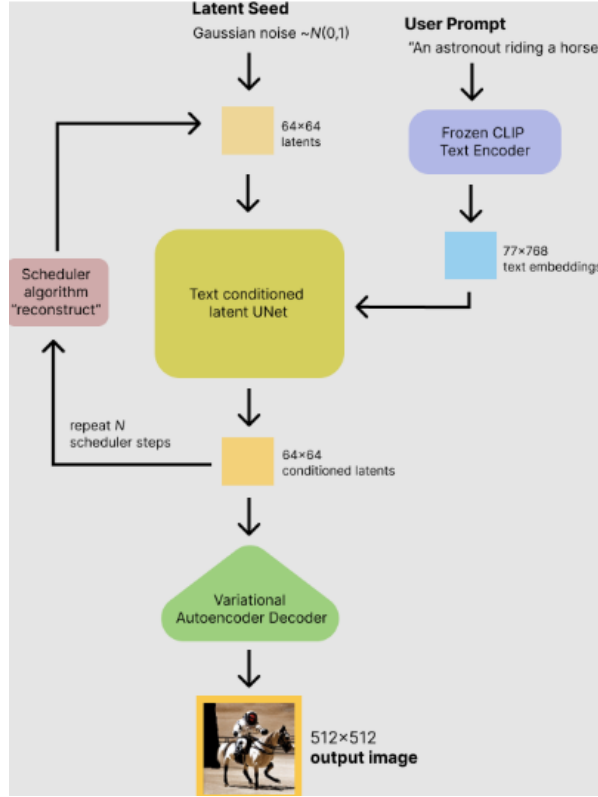


Figura 2.2: Schema esempio di generazione con SD 1.5

2.1.2 Z-Image

Introdotta alla fine del 2025 dal team di ricerca *open-source* Tongyi-MAI, Z-Image [52, 46] rappresenta il secondo pilastro della *pipeline* sperimentale di questa tesi e il primo modello testato sull'infrastruttura HPC. Sviluppato con l'obiettivo di sfidare il paradigma del "calcolo a tutti i costi", Z-Image si posiziona come un modello di fondazione da 6 miliardi di parametri. Pur mantenendo requisiti *hardware* accessibili, l'architettura introduce una rivoluzione geometrica rispetto a Stable Diffusion 1.5, abbandonando la U-Net a favore di un *Diffusion Transformer* (DiT).

La vera peculiarità ingegneristica di Z-Image, descritta nel *paper* ufficiale (Z-Image Team et al., 2025), risiede nell'adozione di un'architettura *Scalable Single-Stream DiT* (S3-DiT).

A differenza delle architetture *dual-stream* che processano il testo e i vettori dell'immagine in due canali isolati per poi fonderli solo alla fine tramite *Cross-Attention*, Z-Image opera un'unificazione radicale a livello di sequenza attraverso tre fasi distinte:

- **Il Text Encoder:** Il condizionamento testuale primario abbandona i vecchi limiti dei modelli CLIP di ridotta dimensionalità. L'intera comprensione linguistica viene affidata a Qwen3-4B, un *Large Language Model* leggero integrato direttamente come *encoder*. Questo modulo permette di allineare istruzioni testuali estremamente complesse e articolate con il contenuto visivo, rimuovendo definitivamente il collo di bottiglia dei 77 *token* massimi.
- **Tokenizzazione visiva (Patchification):** l'immagine latente proveniente dall'Autoencoder viene sezionata in piccoli frammenti bi-dimensionali (*patches*) e linearizzata in una sequenza monodimensionale di *token* visivi.
- **Concatenazione (Single-Stream):** i *token* del testo e i *token* dell'immagine vengono concatenati uno di seguito all'altro, formando un'unica sequenza di *input* unificata.

Questo flusso singolo attraversa i blocchi del *Transformer*, consentendo un'interazione *cross-modale* densa e bidirezionale ad ogni singolo strato della rete. Il testo influenza l'immagine e, simmetricamente, la struttura geometrica emergente dell'immagine guida la comprensione del testo. Questa altissima efficienza parametrica permette di rivaleggiare con modelli commerciali significativamente più grandi.

Il passaggio dalla rete U-Net convoluzionale all'architettura S3-DiT permette di superare i limiti strutturali e geometrici intrinseci a modelli come Stable Diffusion 1.5:

- **Flessibilità della risoluzione spaziale:** non essendo più vincolato dalla rigida griglia geometrica delle convoluzioni, Z-Image supporta nativamente risoluzioni flessibili che spaziano da 512×512

fino a 2048×2048 pixel, adattandosi a qualsiasi rapporto d'aspetto (*aspect ratio*) senza generare sdoppiamenti del soggetto o artefatti strutturali.

- **Coerenza fotorealistica e dettaglio:** la capacità intrinseca dei *Transformer* di calcolare le relazioni di attenzione a lungo raggio si traduce in una resa nettamente superiore dei dettagli ad alta frequenza, fondamentale per la ricostruzione accurata delle *texture* della ceramica, dei reperti metallici e delle fisionomie degli *avatar* storici.

Tuttavia, sebbene i 6B di parametri e la logica *Single-Stream* garantiscano un'eccellente precisione estetica e testuale, le relazioni spaziali microscopiche e la rigidità filologica richiesta da *dataset* archeologici estremamente complessi spingono la sperimentazione verso il gradino successivo, ovvero i modelli della famiglia Flux.

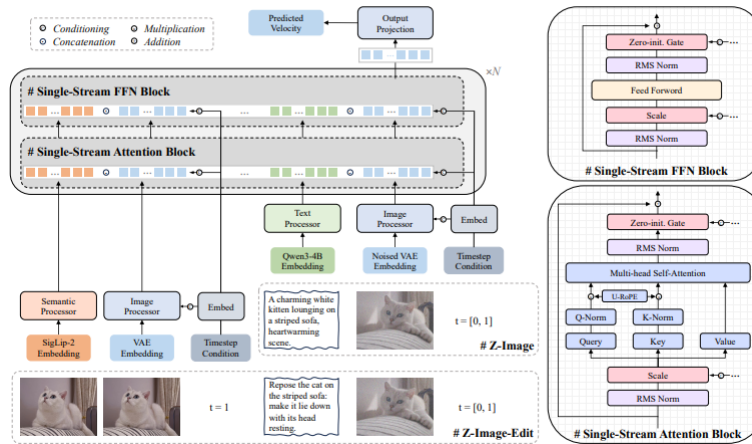


Figura 2.3: Architettura Z-Image

2.1.3 Flux.1

Rilasciato nell'agosto 2024 da Black Forest Labs (BFL), Flux.1 rappresenta lo stato dell'arte dei modelli di generazione d'immagine *open-weights*. Nello specifico, per le attività di tirocinio è stata impiegata

la variante Flux.1 [dev], un modello distillato nella guida (*guidance-distilled*) concepito per scopi di ricerca non commerciale, caratterizzato da un numero di pari a 12 miliardi.

Flux.1 [23, 19] abbandona completamente i paradigmi precedenti unendo l'architettura *Diffusion Transformer* (DiT) alla tecnologia matematica del *Flow Matching* (Esser et al., 2024), un *framework* che ottimizza la generazione sostituendo le traiettorie curve e probabilistiche dei vecchi modelli di diffusione con linee di trasporto dirette e rettilinee (*Rectified Flows*) tra il rumore e l'immagine finale. L'infrastruttura si compone di tre macro-moduli altamente specializzati:

- **Il doppio Text Encoder:** utilizzando due *Text Encoder* si risolve definitivamente la debolezza semantica dei modelli di prima generazione integrando una *pipeline* di condizionamento a due canali paralleli. Il testo in *input* viene processato simultaneamente da CLIP ViT-L/14 e dal più potente *Large Language Model* T5-XXL da ben 4.7 miliardi di parametri. Quest'ultimo agisce come un interprete linguistico profondo, azzerando qualsiasi limite di *token* e permettendo al modello di comprendere descrizioni testuali estremamente articolate, sfumature storiche e relazioni spaziali complesse.
- **Il motore MMDiT (Multi-Modal Diffusion Transformer):** a differenza della logica *Single-Stream* analizzata in Z-Image, Flux.1 adotta un'architettura *Dual-Stream* interna che evolve in *Single-Stream*. Per i primi strati del *Transformer*, i *token* del testo e i *token* dell'immagine vengono processati in due flussi paralleli isolati, dotati di *embedding* posizionali e parametri di modulazione propri. Successivamente, i vettori vengono concatenati in un flusso unificato per calcolare l'attenzione *cross-modale* globale.
- **L'Integrazione delle Parallel Attention e RoPE:** per garantire una stabilità geometrica assoluta ed evitare allucinazioni strutturali, Flux.1 implementa strati di attenzione paralleli e l'algoritmo RoPE (*Rotary Positional Embedding*) a livello multi-modale, anco-

rando matematicamente ogni *patch* dell'immagine alla sua esatta coordinata tridimensionale latente.

Il passaggio alla scala computazionale dei 12B di parametri e la transizione matematica verso i *Rectified Flows* offrono benefici rivoluzionari per gli obiettivi di ricostruzione filologica della tesi:

- **Comprensione terminologica specialistica:** grazie al supporto del modello linguistico T5-XXL, Flux.1 non ignora e non modernizza i termini d'epoca o i tecnicismi archeologici. Il modello decodifica la richiesta nel suo reale contesto storico.
- **Resa del Text Rendering Nativo:** Flux.1 dimostra la capacità di generare scritte, epigrafi e caratteri alfabetici coerenti e privi di distorsioni all'interno del tessuto visivo.
- **Precisione anatomica e proporzionale:** l'interazione profonda garantita dai blocchi MMDiT diminuisce notevolmente gli artefatti geometrici e le duplicazioni dei soggetti. Le fisionomie umane e le simmetrie degli elementi architettonici vengono riprodotte rispettando i rigidi vincoli accademici imposti dai *dataset* di addestramento.

L'esecuzione e il *fine-tuning* tramite moduli LoRA di un'infrastruttura di questa portata sarebbe impraticabile su *hardware consumer* isolato, consolidando definitivamente il ruolo imprescindibile dei supercomputer HPC.

2.1.4 Flux.2

Rilasciato da Black Forest Labs tra la fine del 2025 e i primi mesi del 2026, Flux.2 [4, 2, 3] rappresenta l'evoluzione diretta e l'attuale stato dell'arte della famiglia Flux. Pur appoggiandosi sulle fondamenta matematiche poste dal suo predecessore, il modello introduce ottimizzazioni architettoniche radicali volte a ridefinire il rapporto tra l'allineamento semantico del *prompt* e il consumo di risorse computazionali durante i cicli

di calcolo ad alte prestazioni. Nello specifico della presente trattazione, l'analisi si concentra sulla variante Flux.2 [dev], un modello *open-weights* distillato nella guida che compie presenta un notevole aumento nel numero di parametri, raggiungendo una scala complessiva di 32 miliardi di parametri.

Questo non non è un semplice incremento quantitativo, ma il risultato di una fusione multimodale rivoluzionaria. Flux.2, infatti, integra nativamente un *Vision-Language Model* (VLM) Mistral-3 da 24 miliardi di parametri all'interno del proprio *Rectified Flow Transformer*. Questa sinergia consente al modello di sostituire i paradigmi di codifica testuale isolati delle prime generazioni, permettendo di elaborare simultaneamente la semantica profonda del testo e i vettori latenti geometrici dell'immagine con un'alta densità concettuale.

L'infrastruttura introduce tre innovazioni chiave rispetto alla prima generazione:

- **Architettura Asymmetric MMDiT-2:** la vera rivoluzione strutturale risiede nel superamento della logica *Dual-Stream* simmetrica di Flux.1. Si è notato che, mentre i *token* dell'immagine cambiano dinamicamente ad ogni passo di *denoising*, i *token* semantico-testuali estratti dal modulo di grande scala rimangono statici. Flux.2 implementa quindi un meccanismo asimmetrico. Per cui, dopo i primi strati del *Transformer*, il modello congela il ricalcolo dei vettori linguistici e indirizza il 100% della capacità computazionale dei blocchi successivi esclusivamente sulla generazione e sulla coerenza spaziale delle *patch* visive.
- **Condizionamento e prompt adherence:** grazie alla spina dorsale multimodale e alla sinergia ottimizzata con il doppio *text encoder*, Flux.2 azzera i rari fenomeni di omissione (*prompt dropping*) riscontrati nelle lunghe e dense descrizioni testuali, garantendo che ogni singolo vincolo semantico inserito dall'utente venga rigorosamente tradotto in *pixel*.

- **Ottimizzazione del Campionamento (NFE):** sfruttando un algoritmo di *Flow Matching* di seconda generazione, il modello accelera drasticamente la convergenza dello spazio latente. Il processo di *denoising* si completa in un arco ideale di soli 16-20 passi di inferenza senza alcuna perdita di precisione geometrica.

I miglioramenti architetturali introverti da Flux.2 si traducono immediatamente in un salto qualitativo per gli obiettivi di ricostruzione filologica della ricerca archeologica e culturale:

- **Generazione Ultra-High-Definition nativa:** grazie all'evoluzione degli *embedding* posizionali rotanti all'interno del flusso asimmetrico, Flux.2 supporta nativamente *output* a risoluzioni superiori (fino a 1.5K e 2K). Questo permette di mappare *texture* microscopiche e ricche di dettagli complessi direttamente in fase di prima inferenza, eliminando la necessità di ricorrere a moduli LoRA di *upscaling* esterni che rischiano spesso di alterare i dettagli originali.
- **Perfezionamento dell'epigrafia e del Text Rendering:** Flux.2 eleva la capacità di sintesi dei caratteri alfabetici integrando il testo nella struttura latente.
- **Rigidità Proporzionale e Prospettica:** il controllo asimmetrico del flusso azzera le aberrazioni prospettiche nell'elaborazione di elementi architettonici seriali. Il modello mantiene una coerenza matematica rigorosa tra il primo piano e lo sfondo, rispettando le proporzioni geometriche imposte dai canoni dell'architettura antica.

In definitiva, se Stable Diffusion 1.5 ha rappresentato l'alba della sintesi d'immagine e Z-Image e Flux.1 hanno aperto le porte alla flessibilità dei *Transformer*, Flux.2 consolida questi concetti e li migliora, offrendo possibilità di generation visiva ad elevata precisione e aderenza filologica.

2.2 Tecniche di Fine-Tuning efficiente: Il metodo LoRA (Low-Rank Adaptation)

La tecnica *Low-Rank Adaptation* (LoRA), introdotta da Hu et al. (2021), rappresenta il paradigma di riferimento nell'ambito dei metodi di *Parameter-Efficient Fine-Tuning* (PEFT). Originariamente concepita per i grandi modelli linguistici, è diventata lo standard che permette di adattare un modello di apprendimento automatico di grandi dimensioni per usi specifici senza dover riaddestrare l'intero modello. Usando un LoRA non è necessario stravolgere tutta la sua struttura, ma basta applicare una correzione mirata e leggera inserendo delle coppie di matrici di rango inferiore (*rank-decomposition matrices*) all'interno degli strati della rete neurale, senza toccare i pesi pre-addestrati del modello base.

2.2.1 Fondamenti Matematici e Decomposizione a Basso Rango

In una rete neurale le conoscenze sono immagazzinate in grandi matrici di pesi (W). Quando si vuole adattare un modello, l'approccio tradizionale prevederebbe la modifica di ogni singolo valore dentro W . LoRA ipotizza che l'aggiornamento necessario abbia un rango intrinseco basso, così la matrice di modifica (ΔW) può essere scomposta nel prodotto di due matrici molto più piccole, chiamate A e B :

$$\Delta W = B \cdot A \tag{2.1}$$

Dove le dimensioni spaziali delle nuove matrici sono definite rigorosamente come:

$$A \in \mathbb{R}^{r \times k} \tag{2.2}$$

$$B \in \mathbb{R}^{d \times r} \tag{2.3}$$

Il parametro scalare r rappresenta il *Rank* (rango) della decomposizione. Vincolando la transizione matematica a un valore di r drasticamente inferiore rispetto alle dimensioni originarie ($r \ll d, k$). Il numero di parametri addestrabili subisce un abbattimento quantitativo pur mantenendo intatta la capacità espressiva necessaria a mappare le nuove regolarità geometriche del *dataset*.

La formulazione fondamentale che governa il passaggio del segnale (*forward pass*) attraverso un *layer* modificato con tecnologia LoRA è:

$$h = W_0x + \Delta Wx = W_0x + \frac{\alpha}{r}BAx \quad (2.4)$$

Dove:

- W_0 : è la matrice dei pesi originali del modello che rimane congelata e immutabile.
- A e B : sono le due matrici a rango basso che contengono la nuova informazione.
- x : è l'*input* che attraversa lo strato della rete neurale.
- h : è il vettore delle attivazioni risultanti in *output*.
- α : è il fattore di scala che determina l'intensità e l'impatto geometrico della modifica LoRA rispetto al modello originale.

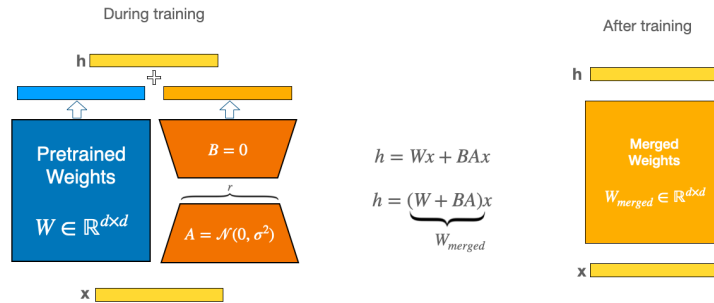


Figura 2.4: Architettura LoRA

2.2.2 Inizializzazione, Dinamica di Scala e Parametri Chiave

Un aspetto cruciale per garantire la stabilità matematica del modello ed evitare il deterioramento delle capacità generative originarie all’inizio del *training* risiede nella strategia di inizializzazione delle matrici di rango ridotto.

Alla partenza del ciclo di addestramento:

- **La matrice A** viene inizializzata seguendo una distribuzione gaussiana casuale, garantendo una varianza controllata del segnale.
- **La matrice B** viene interamente inizializzata a zero.

A causa di questa asimmetria iniziale, il prodotto $\Delta W = B \cdot A$ risulta essere identicamente nullo al *timestep* zero. Di conseguenza, nelle primissime iterazioni del processo di ottimizzazione, la componente LoRA è matematicamente invisibile, permettendo al modello di partire basandosi esclusivamente sulle sue solide competenze native ed eliminando qualsiasi artefatto distruttivo transitorio.

Il controllo del flusso perturbativo e la calibrazione degli *output* sono regolati da due parametri fondamentali che strutturano l’addestramento:

- **Rank (r):** determina l’ampiezza dello spazio latente interno alle matrici LoRA. Un rango estremamente ridotto comprime la modifica isolando esclusivamente i macro-tratti semantici, risultando ideale per l’apprendimento di stili artistici lineari. Un rango più elevato espande la capacità di memoria del modulo, configurandosi come un requisito imprescindibile per immagazzinare dettagli microscopici e vincoli geometrici rigidi ad alta precisione.
- **Alpha:** rappresenta un fattore di scala costante che agisce come un moltiplicatore geometrico. Nella funzione di attivazione, il contributo di ΔW viene pesato attraverso il rapporto tra alpha e r . Durante il processo di discesa del gradiente, questo fattore stabilizza l’addestramento. Quando si modifica il *Rank*, il rapporto di

scala costante evita la necessità di ricalibrare daccapo il tasso di apprendimento, modulando l'intensità e l'indipendenza espressiva del modulo LoRA rispetto al modello base.

2.2.3 Ottimizzazione Computazionale e vantaggi

L'implementazione dell'architettura LoRA offre una serie di vantaggi che abbracciano l'intero ciclo di vita del modello, ottimizzando drasticamente il bilancio tra efficienza di calcolo e accuratezza filologica:

- **Efficienza nell'allocazione della VRAM (Training):** poiché l'algoritmo di *Backpropagation* calcola e aggiorna i gradienti esclusivamente per le matrici A e B , la stragrande maggioranza della struttura neurale non richiede l'allocazione della memoria per gli stati dell'ottimizzatore. Questo decremento riduce drasticamente il *memory footprint* della GPU, consentendo di traslare l'addestramento di modelli complessi e mastodontici su nodi multi-GPU HPC senza incorrere in saturazioni della memoria volatile (*Out of Memory*).
- **Contenimento dello spazio di archiviazione (Storage):** poiché il rango ridotto limita i parametri attivi a una frazione infinitesimale (spesso inferiore allo 0,01% dell'architettura nativa), la dimensione fisica del file finale risulta microscopica. A fronte di modelli base che occupano decine di Gigabyte su disco, il *checkpoint* LoRA si attesta nell'ordine di poche decine o centinaia di Megabyte. Ciò ottimizza la gestione dello *storage* sul supercomputer e agevola lo stoccaggio, il trasferimento rapido e la catalogazione di decine di varianti archeologiche specializzate.
- **Zero Latenza in Fase di Inferenza (Weight Merging):** in fase di generazione l'architettura permette di eseguire l'operazione di *Weight Merging*. Sfruttando la linearità delle operazioni matriciali, la matrice ΔW può essere permanentemente sommata alla matrice

originale W_0 prima dell'esecuzione dei cicli di *denoising*. Questo azzerà qualsiasi *overhead* computazionale o ritardo di calcolo rispetto alle performance del modello nativo.

- **Modularità funzionale e intercambiabilità:** l'indipendenza strutturale trasforma il modello base in un nucleo molto versatile. È possibile disaccoppiare, scambiare o combinare simultaneamente diversi moduli LoRA sul medesimo modello.

2.3 L'ecosistema ComfyUI

Per la progettazione, l'orchestrazione e l'esecuzione dell'intera *pipeline* generativa è stato adottato l'ecosistema ComfyUI [10]. Si configura come un potente *framework* di calcolo modulare basato sulla teoria dei grafi. Nello specifico, l'interfaccia organizza l'architettura dei modelli di diffusione e dei *Transformer* attraverso un modello a Grafi Diretti Aciclici (DAG - *Directed Acyclic Graphs*).

In questa struttura, ogni blocco operativo viene isolato all'interno di un singolo Nodo deputato a una specifica funzione matematica o algoritmica. I nodi sono interconnessi tra loro mediante collegamenti (*edges*) vincolati a tipi di dati rigorosamente tipizzati (modelli, condizionamenti, immagini latenti, pesi d'attenzione). Questo approccio garantisce una trasparenza metodologica assoluta, offrendo all'utente un controllo granulare su ogni singola fase del processo di *denoising* e consentendo l'integrazione flessibile e simultanea di modelli radicalmente differenti all'interno della medesima sessione di calcolo.

2.3.1 Componenti Fondamentali del Grafo Generativo

Sebbene la topologia di un *workflow* possa variare in base alla complessità del compito filologico richiesto, la replica e il controllo delle fasi essenziali

di un Modello di Diffusione Latente (LDM) o di un *Transformer* di Flusso poggiano su una serie di nodi fondamentali standardizzati:

- **Load Checkpoint:** rappresenta l'iniziatore del flusso. Questo nodo è deputato al caricamento in memoria e alla scomposizione del file del modello base, separandone le tre componenti logiche originarie, ovvero l'architettura di *denoising* (U-Net o DiT), il *Text Encoder* e l'Autoencoder.
- **Empty Latent Image:** questo nodo genera un tensore di rumore gaussiano puro all'interno dello spazio latente compresso, definendo le dimensioni spaziali dell'output e la dimensione del *batch* (*batch size*), determinando così i requisiti di memoria allocati preliminarmente sulla GPU.
- **CLIP Text Encode (Prompting):** è il modulo responsabile del condizionamento semantico. Il nodo riceve in *input* le istruzioni testuali in linguaggio naturale, sfruttando i *Text Encoder* associati le converte in *embedding* numerici ad alta dimensionalità. Questi vettori vengono successivamente proiettati all'interno dei meccanismi di *Cross-Attention* del modello di *denoising*.

2.3.2 Il KSampler

Il nucleo computazionale dell'intero *workflow* è rappresentato dal nodo KSampler [9, 39]. Questo blocco esegue il processo iterativo di rimozione del rumore all'interno dello spazio latente, accettando in *input* il modello di *denoising*, i vettori di condizionamento testuale e il rumore latente iniziale.

Il controllo fine dell'estetica, della convergenza e dell'accuratezza scientifica dell'*output* è regolato da una serie di parametri parametrici configurabili internamente:

- **Seed:** definisce il valore numerico del seme di partenza del generatore di numeri pseudo-casuali che determina la disposizione spaziale

del rumore gaussiano iniziale. L'utilizzo di un *seed* fisso garantisce la perfetta riproducibilità matematica dell'esperimento visivo a parità di vincoli collaterali.

- **Steps:** indica il numero di passaggi discreti in cui viene suddivisa l'integrazione numerica della traiettoria di *denoising*. Un incremento degli *steps* tende a rifinire i dettagli ad alta frequenza e a stabilizzare la *micro-texture* dei materiali storici, sebbene determini un aumento lineare del tempo di calcolo.
- **CFG (Classifier-Free Guidance):** regola il coefficiente di aderenza del modello rispetto al condizionamento del *prompt*. Valori ridotti lasciano al modello un maggior grado di libertà creativa nello spazio latente, mentre valori eccessivamente elevati tendono a saturare la gamma cromatica e a irrigidire la geometria, generando artefatti strutturali complessi. Nel caso di modelli basati su *Flow Matching* (come Flux), questo parametro viene sostituito o guidato da nodi di calcolo dedicati alla *Guidance Scale*.
- **Denoise:** determina l'intensità della perturbazione applicata al tensore latente. Un valore pari a 1.0 identifica l'elaborazione di un rumore nativo puro (tipico del flusso *Text-to-Image*), mentre valori inferiori indicano che il processo opera su una base parzialmente strutturata, configurazione cardine per i compiti di *Image-to-Image* e *Inpainting*.

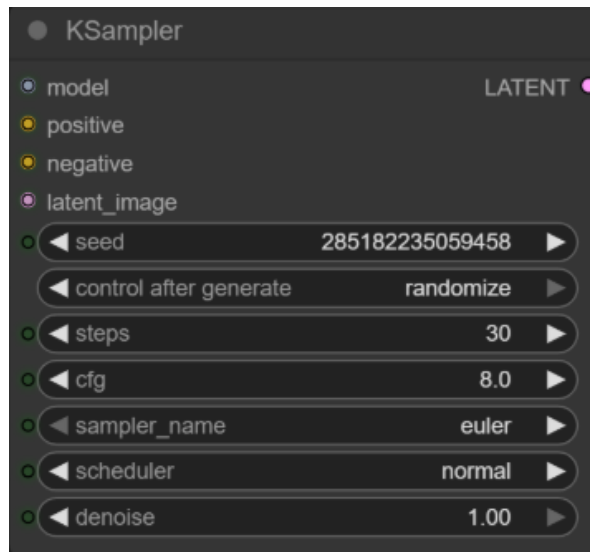


Figura 2.5: Nodo Ksampler

Le variabili a maggiore impatto scientifico inserite nel KSampler sono tuttavia il *Sampler* (campionatore) [45] e lo *Scheduler* (pianificatore) [8]:

- **Il Sampler:** rappresenta l'algoritmo matematico deputato alla risoluzione numerica delle equazioni differenziali (stocastiche o ordinarie) che governano il trasporto del rumore verso la distribuzione dei dati reali. I campionatori si dividono principalmente in deterministici (come i solutori ODE del tipo DDIM o DPM-Solver) e stocastici (solutori SDE che introducono perturbazioni controllate ad ogni iterazione, come i metodi *Ancestral*). Questi ultimi, manifestano una minore prevedibilità geometrica a parità di passi d'inferenza.
- **Lo Scheduler:** controlla la progressione e la tempistica del tasso di abbattimento del rumore lungo l'asse temporale dei passaggi. Esso decide la quantità di rumore da sottrarre ad ogni singolo *step*, modulando la transizione tra le macro-strutture prospettiche iniziali e il microscopico perfezionamento epigrafico o materico finale. Nell'ambito del lavoro svolto, le combinazioni più stabili ed efficienti

hanno previsto l'adozione di algoritmi come Euler e DPMP _2M, associati a mappature temporali di tipo Normal e Karras.

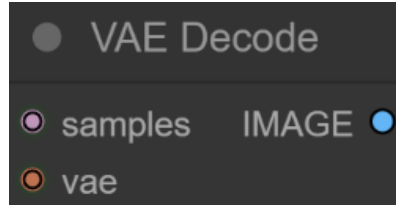


Figura 2.6: Nodo VAE Decode

La chiusura geometrica del grafo è affidata alle funzioni dell'Autoencoder (VAE) [44], l'elemento matematico deputato a fare da ponte tra lo spazio matematico latente e lo spazio geometrico dei pixel:

- **VAE Encode:** esegue il campionamento e la compressione dimensionale di un'immagine reale esterna, riducendone la risoluzione spaziale e traducendola in vettori latenti. Questo passaggio è indispensabile per preparare le immagini di riferimento o le maschere di restauro digitale prima del *training* o dell'inferenza.
- **VAE Decode:** posizionato stabilmente al termine della *pipeline* del grafo, inverte la compressione. Il nodo riceve il tensore latente interamente ripulito dal KSampler e lo decompone proiettandolo nello spazio *pixel* nativo, traducendo la matrice matematica nell'immagine visiva finale del reperto o del personaggio storico ad altissima definizione.

Capitolo 3

Pipeline Sperimentale e Prototipazione Locale

La transizione dai modelli teorici alla loro effettiva applicazione nel contesto filologico-archeologico richiede l'implementazione di una rigorosa pipeline sperimentale preliminare. Prima di traslare l'addestramento su larga scala sull'infrastruttura di calcolo ad alte prestazioni Leonardo, la ricerca ha previsto una fase fondamentale di prototipazione in ambiente locale su hardware consumer.

In questa fase si è voluto, da un lato validare la qualità metodologica, la coerenza geometrica e la densità informativa del dataset proprietario e dall'altro, stabilire una linea di partenza computazionale sfruttando l'architettura classica di Stable Diffusion 1.5, isolando così i limiti strutturali dei modelli convoluzionali di prima generazione per giustificare la migrazione verso le più complesse architetture basate su Transformer.

3.1 L'Ambiente Software di Prototipazione: Kohya_ss

Il framework centrale selezionato per l'esecuzione del fine-tuning preliminare in ambiente locale è Kohya_ss [29]. Questo strumento si configura nel panorama dell'apprendimento automatico come un potente wrapper

grafico e funzionale basato sulla suite open-source **sd-scripts**. L'architettura del software è ingegnerizzata per astrarre la complessità computazionale degli script Python sottostanti, esponendo un'interfaccia web estremamente granulare e intuitiva, pur mantenendo intatta la capacità di manipolazione di basso livello dei parametri di rete.

Una delle caratteristiche metodologiche più rilevanti di Kohya_ss risiede nella sua flessibilità di configurazione. Il software consente infatti di esportare l'intera configurazione parametrica impostata dall'operatore sia in formato JSON (per un rapido ripristino dei vettori tramite interfaccia grafica) che in formato TOML.

L'esportazione in file descrittivi TOML garantisce il principio scientifico della replicabilità sperimentale, per cui l'intera pipeline di addestramento può essere salvata, archiviata e soprattutto invocata direttamente da riga di comando (CLI).

L'interfaccia di Kohya_ss espone una serie di controlli critici che governano la discesa del gradiente e la riconfigurazione dei pesi sinaptici all'interno delle matrici LoRA.

3.1.1 Gestione delle Epoche, dei Batch e delle Dinamiche Temporal

Il campionamento temporale dell'addestramento è regolato dal delicato equilibrio tra epoche e dimensione dei lotti di dati:

Epoca: identifica un ciclo computazionale completo durante il quale l'intero volume del dataset viene presentato alla rete neurale, elaborato in avanti (*forward pass*) e propagato all'indietro per il calcolo dell'errore (*backward pass*).

Dimensione del Batch: determina il numero di campioni multimodali (coppie immagine-testo) elaborati in parallelo dalla GPU prima di procedere all'aggiornamento dei gradienti. L'impiego di una Batch Size superiore a 1 è sistematicamente preferibile all'addestramento su singole immagini. Questo perché calcolando la media dell'errore

di previsione su più vettori simultaneamente, il modello stabilizza la traiettoria di ottimizzazione, migliorando la capacità di generalizzazione dello spazio latente ed evitando che la rete associ il concetto target a pose o contesti specifici di un singolo fotogramma.

3.1.2 Ottimizzazione del Learning Rate e Scheduling

Il Tasso di Apprendimento (Learning Rate) definisce l'ampiezza dello step scalare applicato ai vettori dei pesi durante la prima correzione dell'errore. I parametri che vanno a definire il Learning Rate sono:

LR Scheduler Cosine: è stato implementato uno scheduler sinusoidale di tipo Cosine. Il valore del LR decresce progressivamente seguendo una curva a coseno, rallentando gli aggiornamenti man mano che il modello converge verso la configurazione ottimale dei pesi. Scegliere di mantenere costante il tasso di apprendimento potrebbe causare oscillazioni distruttive attorno ai minimi locali della Loss nelle fasi avanzate del training.

LR Warmup: questa strategia introduce una rampa di accelerazione iniziale [22]. Durante una percentuale definita di step primari, il tasso di apprendimento cresce linearmente partendo da zero fino a raggiungere il valore nominale impostato. Il Warmup si rivela essenziale per stabilizzare i primi passaggi computazionali, impedendo che gradienti iniziali eccessivamente aggressivi provochino divergenze premature o la saturazione distruttiva dello spazio latente.

3.1.3 Efficienza Hardware e Contenimento della VRAM

Per rendere sostenibile il calcolo su GPU di fascia consumer durante la fase di prototipazione, Kohya_ss integra tecniche avanzate di ottimizzazione della memoria:

Mixed Precision (FP16): l'addestramento viene eseguito combinando la precisione standard a 32 bit (Single Precision) con la mezza

precisione a 16 bit (Half Precision) [30]. Le operazioni di memorizzazione dei gradienti e i calcoli delle matrici vengono delegati al formato FP16, riducendo drasticamente l'impatto sulla memoria video e accelerando i tempi di calcolo sui Tensor Core, senza inficiare la stabilità numerica del modello.

Ottimizzatore AdamW 8-bit: sfruttando la quantizzazione a blocchi, questa variante del classico ottimizzatore AdamW [34, 18] memorizza i momenti del gradiente a 8 bit anziché a 32 bit. Tale accorgimento abbassa l'occupazione della VRAM di oltre il 60%, permettendo la gestione di reti LoRA ad alto rango anche su architetture hardware dotate di un quantitativo limitato di memoria grafica dedicata.

3.1.4 Dalla GUI alla Linea di Comando

Sebbene la fase di prototipazione locale si sia avvalsa dell'interfaccia grafica per scopi di monitoraggio visivo e debugging immediato, l'architettura di Kohya_ss poggia nativamente su una suite di script Python indipendenti.

La possibilità di svincolare l'addestramento dalla GUI e di operare direttamente tramite riga di comando rappresenta il prerequisito tecnologico per la successiva migrazione della pipeline sull'infrastruttura di supercalcolo HPC Leonardo, procedura che verrà esplicitata più avanti.

Il flusso logico orchestrato in locale riproduce fedelmente i meccanismi di invocazione che verranno scalati su larga scala. Previa attivazione dell'ambiente virtuale Python e configurazione dell'accelerazione hardware tramite la libreria Hugging Face `accelerate`, il processo viene istruito lanciando il comando:

```
accelerate launch train_network.py --config_file="percorso  
/configurazione.toml"
```

L'algoritmo acquisisce così in modo deterministico la posizione del modello di fondazione nel file system, i puntamenti alle directory del dataset e i vincoli parametrici estratti dal file TOML.

3.2 Ingegnerizzazione e Creazione del Dataset

La qualità espressiva e l'accuratezza filologica di un modulo LoRA dipendono in modo biunivoco dalla struttura del dataset di addestramento. Il dataset non si configura come una mera raccolta di immagini, ma come un'infrastruttura multimodale bilanciata, in cui ad ogni elemento visivo deve corrispondere un'accurata codifica testuale in formato di metadato (`.txt`). Per garantire l'ancoraggio del nuovo concetto nello spazio latente del modello senza sovrascrivere le distribuzioni preesistenti, è stata definita una parola chiave univoca (*trigger word* o *instance token*), inserita come prefisso obbligatorio in ogni file descrittivo, la quale funge da attivatore del modulo LoRA in fase di inferenza.

Per verificare la flessibilità di risposta della pipeline di lavoro sono stati utilizzati soggetti diversi e di diverso tipo: l'avatar storico "Apa", il personaggio centrale "Ati" e un modello architettonico di tempio etrusco. Solo dopo aver creato l'ambiente di lavoro sull'infrastruttura HPC si è utilizzata un'antefissa, in modo da verificare l'utilizzo del lavoro svolto quando il soggetto è oggetto.

3.2.1 Rendering dei Soggetti

A causa della differente natura dei soggetti digitali disponibili, la costruzione dei vari dataset ha richiesto l'adozione di due distinte strategie computazionali e di Computer Grafica (CG):

Apa: essendo il modello 3D originario troppo datato, non è stato possibile operare direttamente su di esso. Si è resa necessaria un'operazione di campionamento ed estrazione dei frame. Sfruttando la

suite multimediale a riga di comando **FFmpeg**, il flusso video è stato decompresso e frammentato in singoli fotogrammi ad alta definizione. Successivamente, è stata eseguita una severa selezione manuale volta a isolare esclusivamente i frame caratterizzati da variazioni significative nella posa, nell'angolazione prospettica e nell'illuminazione, in modo da evitare la ridondanza informativa che avrebbe indotto fenomeni di overfitting.

Ati, Tempio Etrusco, Tomba Etrusca e Antefissa: per questi soggetti si disponeva dei modelli geometrici nativi poligonali. L'estrazione delle immagini è stata orchestrata in modo interamente controllato all'interno dell'ambiente di produzione 3D Blender. Questa metodologia ha permesso di configurare una pipeline di rendering virtuale di precisione geometrica. I soggetti sono stati catturati variando sistematicamente la lunghezza focale della camera, l'altezza dell'orizzonte e l'incidenza dei vettori di illuminazione (*Three-Point Lighting*), garantendo una copertura sferica e multi-angolare del soggetto indispensabile per l'apprendimento delle volumetrie da parte della rete neurale. Per i rendering di Ati e del Tempio, l'illuminazione globale della scena è stata modulata attraverso l'impiego dinamico di svariate mappe di illuminazione HDRI (*High Dynamic Range Imaging*). Al contrario, tale approccio non è stato applicato alla Tomba Etrusca in quanto struttura ipogea chiusa che richiedeva unicamente rendering d'interni con sorgenti luminose dedicate e all'Antefissa, per la quale sono stati eseguiti rendering isolati a 360 gradi del singolo manufatto su fondo neutro. In generale, questa diversificazione degli scenari di luce e delle mappe d'ambiente ha permesso di evitare fenomeni di ridondanza contestuale nel dataset, impedendo alla rete neurale di associare le caratteristiche dello sfondo al soggetto geometrico principale. Il modello è stato indotto a isolare e apprendere esclusivamente la morfologia del soggetto, scongiurando artefatti di fusione semantica tra l'entità e l'ambiente circostante.



Figura 3.1: Esempi di render dei soggetti

Una variazione strategica rispetto alle prime iterazioni del dataset ha riguardato l'integrazione di piani di sfondo (*background planes*) filologicamente accurati direttamente all'interno della scena tridimensionale di Blender. Questa metodologia ha permesso di renderizzare immagini in cui il soggetto era situato nativamente in un contesto storico-ambientale rigoroso. Introducendo tali immagini di contestualizzazione archeologica nel motore di rendering, il modello di diffusione è stato indotto ad associare le caratteristiche dei soggetti al loro contesto. Questa strategia si rivela cruciale in fase di inferenza, dato che il modello viene guidato efficacemente nella generazione di sfondi strutturalmente e filologicamente corretti ogni volta che l'operatore formula un prompt testuale che richiami contesti ambientali simili a quelli strutturati nel dataset di addestramento.

3.2.2 Risoluzione dei Conflitti di Retrocompatibilità dei soggetti

La complessità strutturale del modello tridimensionale dell’avatar “Ati”, caratterizzato da sistemi di rigging e simulazioni fisiche avanzate per la gestione dei movimenti, ha fatto emergere problemi di retrocompatibilità software. Le animazioni originarie erano state programmate su vecchie versioni di Blender, le cui funzioni di calcolo fisico delle collisioni sono state rimosse nelle versioni più recenti.

Nelle versioni precedenti alla 3.x era possibile utilizzare una funzionalità che permetteva di allocare un intervallo di fotogrammi precedenti all’inizio dell’animazione, consentendo alla forza di gravità e alle forze inerziali di stabilizzare le trecce prima del rendering. La sua rimozione ha forzato il calcolo fisico a iniziare in corrispondenza del primo fotogramma t_0 , determinando compenetrazioni e distorsioni geometriche.

Lo script implementato ha superato i limiti della GUI agendo direttamente sui parametri di basso livello del motore fisico. Così facendo è stato possibile abilitare i fotogrammi negativi nelle preferenze globali di sistema e manipolare l’allocazione della memoria della cache fisica. Forzando il calcolo dell’assestamento dinamico nell’intervallo temporale negativo antecedente al frame di avvio, lo script ha garantito il perfetto ripristino dell’integrità anatomica del personaggio e l’automazione del *baking* della fisica su tutti i layer di rendering necessari per il dataset.

3.3 Strategie di Captioning e Analisi della Regolarizzazione

Una volta consolidata la componente visiva, la pipeline ha previsto la generazione del condizionamento testuale parallelo. All’interno della suite di addestramento Kohya_ss, l’operazione è stata automatizzata mediante l’impiego del modulo di interrogazione visiva WD14. Questo modello di classificazione analizza ogni immagine del dataset e decompone il con-

tenuto visivo in una stringa ordinata di tag semantici, separati da virgole. Il processo è regolato da una soglia di confidenza (*probability threshold*), grazie alla quale solo i tag che presentano una probabilità di corrispondenza superiore al valore impostato vengono iniettati nel file `.txt`. Inoltre, tramite la GUI è possibile impostare una parola chiave che viene inserita automaticamente all’inizio del file di testo.

3.3.1 Il Fenomeno del Language Drift e l’Uso delle Class Images

L’introduzione di un nuovo concetto specifico all’interno di una rete pre-addestrata comporta il rischio di *Language Drift* (deriva linguistica) e *Class Corruption* (corruzione della classe) [38]. Quando si addestra un modello su un soggetto specifico (ad esempio Ati) esiste il rischio che la rete neurale inizi ad associare le caratteristiche uniche di Ati all’intera categoria donna o persona. Senza regolarizzazione, il modello potrebbe dimenticare come generare una donna generica, finendo per produrre sempre tratti simili a quelli del dataset di addestramento. Quindi, oltre alla fase di addestramento è possibile impostare anche una fase di regolarizzazione in cui il modello viene addestrato su immagini generiche della classe di appartenenza usando la parola chiave generica della classe. Questo serve a ricordare al modello che esiste una distinzione netta tra il soggetto specifico e la categoria generale a cui appartiene.

Nell’ambito della presente sperimentazione in locale, si è scelto deliberatamente di non avvalersi delle immagini di regolarizzazione. Tale decisione risponde a precisi criteri di ottimizzazione computazionale e bilanciamento dei parametri:

Prevenzione della corruzione tramite captioning: invece di affidarsi alla regolarizzazione passiva, il fenomeno del *Language Drift* è stato contrastato mediante un captioning meticoloso e approfondito. Descrivendo esplicitamente nei file `.txt` ogni singolo elemento costante dell’immagine, la rete impara a isolare tali dettagli legan-

doli esclusivamente alla *trigger word*, evitando che essi vadano a inquinare i pesi associati alla classe generica.

Contenimento dell’overhead computazionale: l’attivazione delle immagini di regolarizzazione comporta una duplicazione esatta del carico di calcolo, poiché raddoppia il numero di step totali necessari per completare un’epoca. L’esclusione della regolarizzazione ha permesso di salvaguardare la VRAM e di accelerare i tempi di inferenza preliminare senza compromettere la stabilità del modello base.

3.4 Dinamica delle Ripetizioni ed Equazioni di Training

Un parametro cruciale nella fase di preparazione del dataset è il numero di Ripetizioni (*Repeats*) assegnato ad ogni immagine. All’interno di Kohya_ss, le *Repeats* stabiliscono quante volte ogni singola immagine debba essere presentata alla rete neurale all’interno di una singola epoca di addestramento.

Il software acquisisce questo parametro tramite una convenzione di nomenclatura rigida applicata alla struttura delle directory, in cui il nome della cartella sorgente deve essere formattato inserendo come prefisso il numero intero delle ripetizioni, seguito da un carattere separatore (es. `15_ati`). In alternativa, per workflow più complessi o con più soggetti, il valore può essere mappato all’interno di un file di configurazione strutturato in formato TOML.

La corretta calibrazione delle ripetizioni è legata matematicamente alla determinazione degli Step Totali di Training, regolata dalla seguente equazione fondamentale:

$$\text{Total Steps} = \frac{\text{Numero Immagini} \times \text{Repeats} \times \text{Epoche}}{\text{Batch Size}} \quad (3.1)$$

Dove la *Batch Size* identifica il numero di token visivi elaborati parallelamente dalla GPU in un singolo passaggio di calcolo. Un'errata pianificazione di questo equilibrio matematico può condurre a due scenari opposti, entrambi scorretti. Un numero troppo basso di ripetizioni genera un modello underfitted, incapace di riprodurre le rigidzze filologiche dei reperti. Al contrario, un valore eccessivo satura i gradienti, provocando la distruzione dello spazio latente e l'impossibilità per il modello di generare varianti flessibili del soggetto appreso.

3.5 Training Preliminare in Locale: Configurazione di Kohya_ss

Una volta consolidata l'infrastruttura del dataset e definita la strategia di codifica semantica, si è proceduto all'esecuzione del processo di addestramento preliminare.

L'obiettivo di questa sessione locale è stato definire una configurazione ottimale dei parametri per il modello Stable Diffusion 1.5. Configurando l'architettura LoRA, il calcolo è stato indirizzato esclusivamente verso la perturbazione dei pesi dei moduli di attenzione (*Cross-Attention Layers*) della U-Net, mantenendo completamente congelati sia il Text Encoder (CLIP) sia i restanti parametri della rete neurale. Questa scelta strategica ha permesso di salvaguardare le risorse hardware locali (VRAM) e, al contempo, di preservare l'integrità dei token linguistici originali del modello di partenza.

3.5.1 Configurazione dei parametri di Rete

Per l'addestramento focalizzato sulla morfologia dell'avatar "Ati", composto da un dataset di 70 immagini e 8 ripetizioni, sono stati definiti i seguenti parametri di computazione:

Ottimizzatore: è stato impiegato l'algoritmo AdamW, configurato con una precisione di calcolo in virgola mobile a mezza precisione FP16 per ottimizzare l'allocazione della memoria sulla GPU consumer.

Learning Rate: il tasso di apprendimento applicato sia a livello globale sia alla componente della U-Net è stato impostato a 1×10^{-4} , un valore standard e collaudato nella letteratura scientifica per l'ancoraggio di concetti strutturali stabili. Di contro, il tasso di apprendimento del Text Encoder è stato impostato rigidamente a 0. Questa scelta metodologica ha permesso di congelare completamente i pesi del codificatore del testo, impedendo l'alterazione dei token linguistici originari e preservando l'integrità semantica del modello base.

Network Rank e Network Alpha: il rango della matrice di decomposizione (r) è stato configurato a un valore di 32, dimensione ottimale per bilanciare l'immagazzinamento dei dettagli geometrici essenziali e l'efficienza computazionale in locale. Il fattore di scala della regolarizzazione dei pesi (α) è stato impostato a 16, rispettando la prassi ingegneristica che prevede un rapporto di scaling pari alla metà del rango per stabilizzare i gradienti durante la retropropagazione dell'errore.

Batch Size e pianificazione temporale: la dimensione del batch è stata impostata a 2, elaborando due coppie multimodali per singolo passo. Il processo è stato distribuito su un arco temporale di 10 epoche computazionali.

In base ai parametri impostati si può ottenere il numero di step per epoca e di step totali:

$$\text{Step per Epoca} = \frac{70 \text{ immagini} \times 8 \text{ ripetizioni}}{2 \text{ (Batch Size)}} = 280 \text{ step} \quad (3.2)$$

$$\text{Total Steps} = 280 \text{ step} \times 10 \text{ epoche} = 2800 \text{ step totali} \quad (3.3)$$

La sessione di training locale ha richiesto un tempo di calcolo complessivo di circa 50 minuti per giungere a completamento.

3.6 Analisi Quantitativa e Qualitativa dei Risultati Locali

La valutazione dell'efficacia dell'addestramento ha previsto un duplice approccio. Un'analisi quantitativa, basata sul monitoraggio matematico della funzione di perdita (*Objective Function*) e una successiva validazione qualitativa empirica condotta mediante l'ambiente a nodi ComfyUI.

3.6.1 Analisi della Funzione di Perdita e delle Epoche

Il parametro fondamentale per tracciare la convergenza della rete neurale è la *Average Loss*. Quantifica lo scostamento quadratico medio tra il rumore campionato dal modello e quello effettivamente predetto durante la fase di diffusione inversa.

Un valore alto (es. 0.8) indica che il modello non ha ancora appreso i tratti distintivi del soggetto. Invece, valori eccessivamente bassi (sotto 0.05) indicano che il modello sta memorizzando i dettagli delle foto anziché imparare il concetto, rendendo il LoRA rigido e di difficile da usare. In genere, valori ottimali per Stable Diffusion 1.5 sono quelli compresi tra 0.05 e 0.15.

Al termine di ogni epoca computazionale, Kohya_ss ha esportato un checkpoint progressivo del file `.safetensors`, associando ad esso il rispettivo valore di Loss media. L'andamento matematico rilevato è sintetizzato nella tabella seguente:

Epoca	Average Loss	Note
Epoca 1	0.147	Inizio <i>training</i>
Epoca 2	0.153	Oscillazione iniziale
Epoca 3	0.146	Stabilizzazione
Epoca 4	0.143	Lieve miglioramento
Epoca 5	0.125	Minimo matematico
Epoca 6	0.136	Stabilizzazione post-minimo
Epoca 7	0.135	Mantenimento
Epoca 8	0.131	Leggera flessione positiva
Epoca 9	0.137	Saturazione
Epoca 10	0.133	Valore finale del modello

Come evidenziato dall'analisi quantitativa, la funzione di perdita ha manifestato una traiettoria estremamente stabile, oscillando linearmente senza mai scendere al di sotto del valore di guardia di 0.05. Tale dinamica attesta che il modello ha evitato la memorizzazione meccanica dei pixel, preservando la flessibilità dello spazio latente.

Il punto di minimo matematico è stato intercettato in corrispondenza dell'Epoca 5 (Loss = 0.133), mentre l'Epoca 10 ha registrato una risalita infinitesimale (Loss = 0.141, con uno scarto trascurabile di appena 0.008). Una variazione così esigua implica che lo scostamento numerico non definisce univocamente la qualità dell'output.

3.6.2 Valutazione degli Output Grafici in ComfyUI

Per verificare il comportamento dei pesi addestrati, i moduli dell'Epoca 5 e dell'Epoca 10 sono stati importati all'interno di un workflow in ComfyUI, effettuando test di generazione comparativi a parità di seed di generazione e parametri di campionamento.

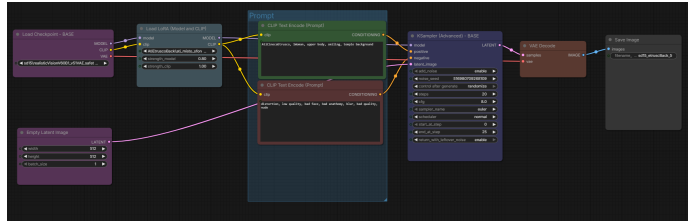


Figura 3.2: Workflow per Stable Diffusion 1.5

In Figura 3.3 è possibile osservare due esempi di generazione, uno effettuato utilizzando il file dell'epoca 5 e l'altro dell'epoca 10.

Il file `.safetensors` dell'Epoca 5 esibisce una spiccata flessibilità combinatoria e una superiore fluidità creativa. Lo spazio latente risponde prontamente a variazioni stilistiche del prompt, pur mantenendo riconoscibili i tratti di Ati.

Il file `.safetensors` dell'Epoca 10 massimizza l'ancoraggio e la somiglianza filologico-estetica con i rendering geometrici originari di Blender, manifestando tuttavia una leggera rigidità maggiore che limita la libertà d'azione dell'operatore nell'introdurre variazioni compositive estreme.



Figura 3.3: Esempi di generazione effettuati utilizzando l'epoca 5 (sinistra) e l'epoca 10 (destra).

Per ottenere risultati di buona qualità come quelli mostrati in Figura 3.3 è stato necessario adottare un modello base addestrato per l'ultra realismo (*Realistic Vision V6.0 B1*), dato che con modelli più comuni si sono ottenuti risultati di qualità inferiore.

3.7 Limiti Strutturali e Analisi Combinatoria (Multi-LoRA)

Un'istanza cruciale della sperimentazione archeologica ha riguardato la combinazione sequenziale di più moduli LoRA all'interno del medesimo processo di diffusione. Nello specifico, si è tentato di sintetizzare un'immagine complessa utilizzando sia il modulo dedicato al personaggio di "Ati" (LoRA 1) che quello del Tempio Etrusco (LoRA 2), al fine di collocare l'avatar all'interno dello scenario architettonico poligonale.



Figura 3.4: Esempio di output generato tramite configurazione ad innesto Multi-LoRA.

3.7.1 Il Fenomeno della Saturazione Latente e dell'Interferenza dei Pesì

L'interazione simultanea di moduli moltiplicativi introduce severi problemi di bilanciamento parametrico. L'applicazione di più LoRA equivale matematicamente a iniettare vettori di perturbazione multipli sulle medesime matrici di proiezione della *Cross-Attention* originale:

$$W_{\text{finale}} = W_0 + \left(\frac{\alpha_1}{r_1}\right) B_1 A_1 + \left(\frac{\alpha_2}{r_2}\right) B_2 A_2 \quad (3.4)$$

I test empirici condotti variando i moltiplicatori di forza (*weights*) in ComfyUI hanno evidenziato i limiti strutturali di Stable Diffusion 1.5:

Interferenza semantica: impostando pesi elevati su entrambi i moduli, le matrici BA si influenzano a vicenda. Il modello subisce un collasso parziale della coerenza visiva, generando artefatti strutturali o alterando l'anatomia del personaggio in funzione delle geometrie del tempio.

Insufficiente densità informativa: riducendo i pesi per evitare la sovrapposizione, il livello di dettaglio macro-geometrico decade drasticamente, restituendo immagini imprecise e prive del rigore filologico richiesto dalla disciplina archeologica.

Questo limite non è imputabile a un'errata calibrazione del dataset, ma costituisce un vincolo architettuale invalicabile della U-Net di SD 1.5, il cui spazio latente compresso (basato su una risoluzione nativa di soli 512×512 pixel) possiede una capacità e una larghezza di banda informative troppo ridotte per gestire il condizionamento contemporaneo di prompt complessi e matrici multi-rango sovrapposte.

3.7.2 Validazione dei Moduli Singoli e Strategie di Mitigazione

Al contrario, isolando i moduli ed eliminando la sovrapposizione dei pesi, le prestazioni locali si sono rivelate del tutto congruenti con le sorgenti. La generazione focalizzata esclusivamente sull'architettura del Tempio Etrusco (Figura 3.5) ha dimostrato una perfetta corrispondenza geometrica e formale con la mesh tridimensionale di Blender utilizzata per l'addestramento.



Figura 3.5: Validazione e resa geometrica del modulo LoRA singolo applicato al Tempio Etrusco.

Allo stesso modo, sono stati riscontrati limiti di rendering della figura intera sul personaggio di Ati. Le inquadrature a mezzo busto o in primo piano risultano definite, mentre i dettagli del volto decadono vistosamente nelle inquadrature a figura intera.

Questa problematica è stata parzialmente mitigata in locale grazie all'integrazione dei piani di sfondo filologicamente corretti direttamente nel rendering delle immagini di addestramento, come spiegato nel paragrafo 2.1.1. Avendo pre-condizionato la rete a riconoscere l'orizzonte archeologico come estensione naturale del personaggio, si è alleggerito il carico di decodifica semantica richiesto al prompt in fase di inferenza.

L'impossibilità di scalare la risoluzione spaziale oltre i confini fisici della U-Net e l'insorgenza di artefatti nella combinazione Multi-LoRA hanno decretato la conclusione della fase di prototipazione locale, formalizzando la necessità scientifica di migrare l'intera pipeline di lavoro verso l'infrastruttura di supercalcolo HPC Leonardo per l'addestramento sui modelli di nuova generazione basati su Transformer (DiT).

Capitolo 4

L'infrastruttura HPC

4.1 L'Accesso a Leonardo e l'Infrastruttura di Calcolo

Il supercomputer Leonardo, situato presso il data center del CINECA a Bologna, rappresenta una delle infrastrutture di High-Performance Computing (HPC) più potenti al mondo. L'architettura del cluster è ottimizzata per l'elaborazione di massicci carichi di lavoro paralleli e per l'addestramento di *Deep Neural Network* (DNN) su larga scala, grazie a una dotazione hardware basata su nodi di calcolo equipaggiati con acceleratori grafici NVIDIA Tensor Core ad altissima larghezza di banda.

4.1.1 Nodi di Login e Politiche di Accesso

L'accesso al sistema avviene esclusivamente da remoto tramite connessione protetta sotto protocollo SSH (Secure Shell), previa autorizzazione e autenticazione basata su chiavi crittografiche asimmetriche configurate sulla piattaforma CINECA. L'utente può accedere a dei nodi di login (*login nodes*).

Questi nodi dispongono di connettività internet verso l'esterno e sono dedicate esclusivamente ad attività a basso impatto computazionale, quali la navigazione nel file system, la compilazione del codice, l'or-

ganizzazione delle directory di storage e il trasferimento dei dati (tramite utility come `wget` o `scp`). Su questi nodi non vengono lanciati script di addestramento o di calcolo intensivo, in quanto un sovraccarico comprometterebbe l'accesso al cluster per gli altri utenti.

4.1.2 Caratteristiche Architetture dei Nodi GPU

La pipeline sperimentale di fine-tuning su modelli a parametri massivi richiede un'infrastruttura in grado di sostenere non solo calcoli tensoriali complessi, ma anche una gestione della memoria volatile per evitare colli di bottiglia nel caricamento dei checkpoint. L'esecuzione dei job su Leonardo ha sfruttato la partizione Booster [6].

Sotto il profilo hardware, ciascun nodo di calcolo del modulo Booster è strutturato secondo una configurazione enterprise simmetrica:

- **Accelerazione grafica:** ogni nodo ospita 4 GPU NVIDIA A100 basate su architettura Ampere, ciascuna dotata di 64 GB di memoria HBM2e dedicata ad altissima larghezza di banda (1.6 TB/s). Questo ha permesso di superare i rigidi vincoli di memoria video (VRAM) riscontrati nella prototipazione locale, consentendo l'allocazione in parallelo di batch size elevati e text encoder pesanti come il T5-XXL.
- **Memoria di sistema (RAM):** ciascun nodo di calcolo implementa un tetto massimo di 512 GB di RAM di sistema (DDR4). In fase di sottomissione del job tramite la sintassi di Chappyner, l'allocazione è stata configurata per saturare l'intera disponibilità di memoria del nodo (fino alla soglia limite dei 512 GB). Questa saturazione si è resa indispensabile per gestire le operazioni offline di caching dei tensori grafici e la decompressione simultanea dei dataset multimodali direttamente nella memoria volatile del cluster, azzerando i tempi di latenza legati all'I/O del disco.
- **Sottosistema di rete:** l'interconnessione tra i nodi è garantita da schede NVIDIA InfiniBand HDR che assicurano una latenza mini-

ma nel trasferimento degli asset operativi e preparano l'ambiente a potenziali scalabilità in modalità Multi-GPU distribuita.

4.1.3 Nodi di Calcolo e Gestione delle Code via Slurm

L'esecuzione vera e propria della pipeline di addestramento e dei processi software intensivi è demandata ai nodi di calcolo (*compute nodes*). Tali nodi operano in uno stato di totale isolamento dalla rete internet esterna e le loro risorse hardware vengono allocate in modo deterministico e temporizzato.

La sottomissione dei lavori e la prenotazione dell'hardware avvengono attraverso il sistema di accodamento e gestione delle risorse Slurm (Simple Linux Utility for Resource Management). L'operatore crea uno script contenente le istruzioni specifiche per il lavoro e lo instrada nel sistema tramite il comando `sbatch`. Lo script definisce preventivamente i requisiti necessari:

- La partizione di coda corretta in base alla priorità (Quality of Service o QoS).
- Il quantitativo esatto di GPU NVIDIA da dedicare all'addestramento.
- Il limite di tempo massimo di esecuzione del job (*walltime*).

Una volta che Slurm approva la richiesta e alloca le risorse, il job viene eseguito in modalità background non presidiata, salvando ogni riga di output o di errore all'interno di un file di log testuale riproducibile. Superare i limiti di isolamento di questa architettura per abilitare flussi interattivi e captioning evoluti ha richiesto l'ingegnerizzazione di canali software dedicati, descritti nei paragrafi successivi.

Tuttavia, la metodologia di accesso ai nodi di calcolo utilizzata in questa tesi è stata quella tramite interfaccia Web. Questo è stato possibile grazie all'impiego di **Chappyner** nella filiera di lavoro.

4.2 Orchestrazione Remota e Servizi Web: Chappyner

L'interazione con i nodi di calcolo di un'infrastruttura HPC come il supercomputer Leonardo avviene tradizionalmente in modalità *headless*, ovvero tramite terminale a riga di comando via protocollo SSH e script di accodamento Slurm. Tuttavia, la necessità di monitorare visivamente i flussi di lavoro, debuggare i nodi custom di ComfyUI e verificare in tempo reale i primi output grafici ha richiesto l'implementazione di un canale di comunicazione interattivo bidirezionale.

Per superare i limiti delle architetture tradizionali che richiederebbero l'installazione e la configurazione permanente di demoni Web come Apache o Nginx sui nodi di login, si è scelto di adottare Chappyner [7]. Questo strumento consente di pianificare, instradare ed eseguire servizi Web direttamente sui nodi di elaborazione remoti, rendendoli accessibili dal browser del computer locale attraverso la creazione automatica di un tunnel SSH sicuro.

I servizi di backend su Leonardo vengono isolati e coordinati all'interno di container Apptainer (precedentemente noti come Singularity), garantendo l'immutabilità dell'ambiente software. Sotto il profilo architetturale, Chappyner si compone di tre macro-elementi sinergici:

- **pip_package**: il nucleo software che ospita il codice sorgente del framework e i moduli client deputati alla gestione delle richieste.
- **warehouses**: la directory remota che archivia le implementazioni pratiche e gli script di interfacciamento lato supercomputer.
- **factory**: un sottomodulo critico che raccoglie i file di definizione originari di Apptainer/Singularity. Finché i container rispettano i vincoli strutturali della warehouse, l'operatore può integrare o sostituire i file di definizione con compilazioni personalizzate.

In concreto, una volta configurato l'ambiente virtuale sul cluster, l'allocazione delle risorse e l'avvio del servizio interattivo (ad esempio in

modalità VNC) vengono istruiti tramite una singola chiamata sintattica da terminale, specificando i vincoli di calcolo e le code di priorità (Quality of Service):

```
1 chappyner --user <nome_utente> --cluster leonardo --tool  
   vnc --q <tipo_di_coda> --gres gpu:<n_gpu> --time hh:mm:  
   ss
```

Listing 4.1: Comando di inizializzazione servizio tramite Chappyner

4.3 Evoluzione degli Strumenti di Training: da sd-scripts a musubi-tuner

Il trasferimento della pipeline di fine-tuning su ambiente HPC ha imposto una profonda revisione dei framework software utilizzati in locale. L'installazione di Kohya_ss su Leonardo ha previsto lo sradicamento dell'interfaccia grafica a favore dell'utilizzo nativo e diretto della suite di script a riga di comando contenuta nella cartella **sd-scripts**.

La configurazione dell'ambiente ha richiesto il caricamento deterministico dei moduli software d'architettura distribuiti sul supercomputer CINECA. Tramite il sistema di gestione ambientale del cluster, sono stati inizializzati l'interprete Python 3.11.7 e il toolkit di calcolo parallelo CUDA 12.6:

```
1 module load python/3.11.7  
2 module load cuda/12.6
```

Listing 4.2: Caricamento dei moduli ambientali su Leonardo

Successivamente alla clonazione del repository git, si è proceduto alla creazione di un ambiente virtuale isolato. In questa fase è insorta una criticità legata alla compatibilità dei pacchetti. Per garantire che le librerie di deep learning riconoscessero correttamente i Tensor Core delle GPU NVIDIA di Leonardo è stata eseguita l'installazione manuale e prioritaria del framework PyTorch, vincolandolo alla versione di CUDA attiva,

prima di procedere alla compilazione automatica dei restanti requisiti tramite il file `requirements.txt`.

Al fine di automatizzare questa catena sequenziale di caricamento moduli, attivazione dell'ambiente virtuale e configurazione delle variabili d'ambiente ad ogni nuovo accesso al nodo di calcolo, è stato sviluppato uno script di sottomissione personalizzato denominato `configurazione.sh`.

4.3.1 I Limiti di `sd-scripts` e la Transizione a `musubi-tuner`

Sebbene l'architettura classica di `sd-scripts` si sia rivelata pienamente compatibile con modelli strutturalmente complessi come HunyuanDiT, Stable Diffusion XL (SDXL) e Stable Diffusion 3 e persino con le prime implementazioni della famiglia Flux.1, la rapidissima evoluzione del panorama dei modelli generativi ha evidenziato i limiti di flessibilità di questo ecosistema. Architetture ancor più recenti, imponenti e customizzate basate su Transformer a flusso congiunto, come Flux.2 e i modelli Z-Image, non trovano infatti supporto all'interno della struttura logica originaria sviluppata da Kohya per i modelli commerciali standard.

Per colmare questo gap tecnologico, lo stesso autore di `sd-scripts` ha sviluppato `musubi-tuner` [27], un framework di nuova concezione espressamente ingegnerizzato per scalare il fine-tuning sui modelli basati su Transformer ad altissimo numero di parametri. L'installazione di `musubi-tuner` su Leonardo segue la medesima progressione logica implementata per gli script classici, ma introduce l'integrazione di librerie specializzate di livello enterprise per la tokenizzazione avanzata dei prompt complessi e la manipolazione flessibile dei tensori grafici ad alta densità.

4.4 Ingegnerizzazione dell'Ambiente di Calcolo Offline

Operare all'interno di un'infrastruttura di supercalcolo come Leonardo impone un rigido cambio di paradigma nella gestione delle dipendenze e delle risorse digitali. Come già detto, i nodi di calcolo operano in uno stato di totale isolamento dalla rete internet esterna. Questo determina la necessità di predisporre preventivamente offline ogni singolo file, peso sinaptico o risorsa di configurazione che gli script di addestramento richiamerebbero automaticamente a runtime.

Il processo di configurazione dell'ambiente è stato condotto attraverso una rigorosa metodologia di *trial-and-error*, analizzando sistematicamente i log d'errore per intercettare e risolvere i blocchi causati da chiamate di rete fallite.

4.4.1 Strategie di Trasferimento Asset e Risoluzione dei Conflitti dei Tokenizer

Il primo pilastro del setup ha riguardato l'allocazione dei modelli di fondazione (*base models*) necessari per il fine-tuning dei moduli LoRA. Per il trasferimento dei pesi nell'area di storage del cluster sono state adottate due distinte strategie computazionali:

- **Trasferimento diretto via wget:** eseguito sfruttando la connettività di rete del nodo di login di Leonardo. Questa soluzione si è rivelata l'opzione più efficiente, capitalizzando sulla larghezza di banda dell'infrastruttura CINECA per scaricare i checkpoint direttamente dai repository ufficiali.
- **Trasferimento di rete sicuro via scp:** impiegato come protocollo secondario per migrare dal file system locale al cluster gli asset non reperibili online o pre-elaborati in Blender.

Una criticità architetturale è emersa durante l'invocazione dello script `train_network.py` in relazione alla gestione dei tokenizer, ovvero i mo-

duli software deputati alla conversione del prompt testuale in vettori di embedding discreti. Nonostante il percorso locale del tokenizer venisse esplicitamente mappato come parametro di lancio nel file di configurazione, lo script tentava sistematicamente di inizializzare una connessione HTTP verso i server di Hugging Face per verificare l'integrità del modello linguistico, determinando il crash immediato del job sul nodo di calcolo.

Per aggirare tale vincolo, l'analisi dei log ha permesso di mappare l'esatta gerarchia di memorizzazione attesa dalla libreria `transformers`. È stata pertanto replicata manualmente, all'interno della directory principale di addestramento, la struttura nativa delle sottocartelle del cache storage di Hugging Face, iniettandovi i file di configurazione del Text Encoder. Ingannando l'algoritmo di controllo, lo script ha intercettato i file locali nel percorso atteso, consentendo l'avvio della pipeline di calcolo offline.

4.5 Strategie di Captioning su HPC: Tagging vs Prosa

Il consolidamento del dataset multimodale su ambiente HPC ha richiesto una profonda diversificazione delle strategie di condizionamento testuale. L'insorgenza di architetture generative radicalmente distinte (i classici modelli basati su U-Net e i più recenti modelli basati su Transformer) ha imposto lo sdoppiamento delle tecniche di annotazione semantica delle immagini.

4.5.1 Captioning Discreto: WD14 e l'Ottimizzazione ONNX Runtime

Per i modelli conformi agli standard di Stable Diffusion, la pipeline ha previsto la generazione di metadati in formato tag discreti separati da virgole. Per automatizzare tale processo direttamente sui nodi di calcolo di Leonardo, è stato impiegato lo script di interrogazione visiva WD14.

L'esecuzione offline di questo classificatore ha richiesto l'installazione manuale nell'ambiente virtuale delle librerie di accelerazione hardware `onnx` e `onnxruntime_gpu`. Questo accorgimento ha permesso di convertire i pesi del modello WD14 nel formato standardizzato ONNX, consentendo l'esecuzione delle inferenze di classificazione direttamente sui Tensor Core delle GPU del supercomputer, evitando qualsiasi chiamata ai server remoti e accelerando drasticamente i tempi di generazione dei file `.txt` paralleli al dataset visivo.

4.5.2 Vision-Language Models per Architetture Transformer: Florence-2

Un'inversione di rotta metodologica si è resa necessaria con l'introduzione di modelli di nuova generazione come la famiglia Flux. Queste architetture, basate su blocchi di Transformer a flusso congiunto, non sono progettate per interpretare stringhe di tag atomici, bensì richiedono descrizioni testuale in prosa fluida e verbosa per poter mappare correttamente i concetti nello spazio latente.

A causa delle severe restrizioni e delle complesse catene di dipendenze software necessarie per compilare i modelli di descrizione testuale classici direttamente sui nodi HPC, si è scelto di implementare una strategia di mitigazione flessibile basata sull'impiego di Florence-2 [51, 31], un Vision-Language Model (VLM) d'avanguardia sviluppato da Microsoft.

L'orchestrazione di Florence-2 è stata demandata all'ambiente a nodi ComfyUI, grazie all'utilizzo di appositi *custom nodes*. Utilizzando i *custom nodes* di Florence-2 è possibile selezionare quale modello si vuole utilizzare, se quello base (più leggero e meno potente) o quello large (più pesante ma potente e prolisso). Normalmente verrebbe scaricato a tempo di esecuzione, ma dovendo operare sul nodo di calcolo, è necessario scaricarlo dal nodo di login. Dando in input il percorso della cartella contenente le immagini, queste vengono analizzate una alla volta andando a creare i relativi file testuali. Per il corretto funzionamento del flusso di

lavoro, è sufficiente impostare un numero di job di lancio del workflow pari al numero di immagini.

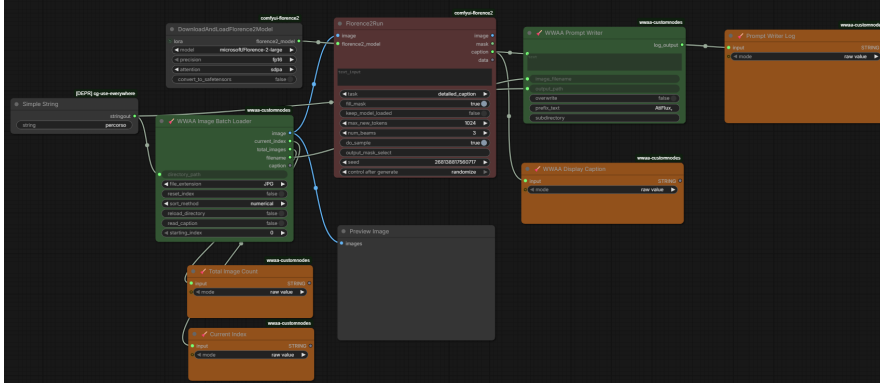


Figura 4.1: Workflow di ComfyUI: Florence-2 per il captioning

Ad ogni iterazione, Florence-2 decodifica le caratteristiche morfologiche del manufatto o dell'avatar, descrivendole più o meno nel dettaglio in base al modello scelto. Il workflow si fa carico di salvare automaticamente l'output sotto forma di file descrittivo `.txt`, applicando la medesima nomenclatura del file d'immagine sorgente e iniettando come prefisso in ogni file la *trigger word* configurata nel workflow stesso.

Questa strategia ha consentito di strutturare dataset filologicamente inattaccabili per i modelli Transformer, superando i limiti di isolamento della rete dell'infrastruttura HPC.

Capitolo 5

Addestramento su Leonardo e Analisi Sperimentale

5.1 Ingegnerizzazione del Training

Il passaggio dalle pipeline di prototipazione locale all'esecuzione di job di addestramento su larga scala all'interno del supercomputer Leonardo ha imposto una riconfigurazione radicale delle metodologie di sottomissione e di orchestrazione dei dati. L'obiettivo primario è stato l'allineamento dei flussi di lavoro ai vincoli strutturali dell'ambiente di High-Performance Computing, massimizzando al contempo il parallelismo computazionale e garantendo la riproducibilità degli esperimenti.

La Figura 5.1 riassume visivamente la pipeline di lavoro di questa fase.

5.1.1 Logica di Controllo a Runtime

Al fine di strutturare il processo di fine-tuning, sono stati realizzati dei file di configurazione in formato TOML per l'astrazione dei parametri [28]. Questa architettura prevede la separazione netta dei domini operativi in due moduli:

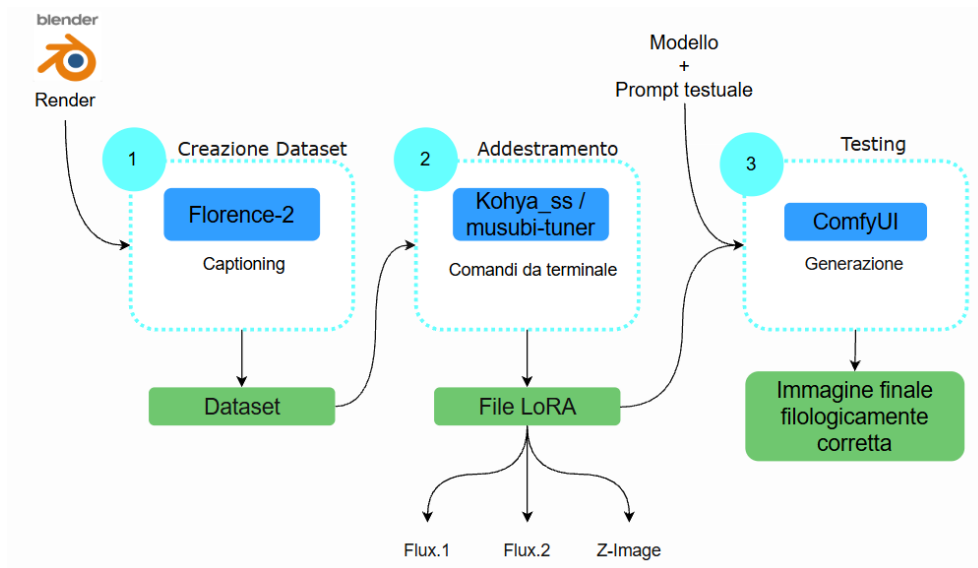


Figura 5.1: Flusso della pipeline di addestramento distribuito e inferenza su infrastruttura HPC Leonardo.

- `dataset_config.toml`: deputato a mappare i puntamenti assoluti alle directory contenenti i file multimediali, a gestire i metadati generati dalle pipeline di captioning e a definire il valore delle ripetizioni per ciascuna classe semantica.
- `training_config.toml`: progettato per fissare i parametri di addestramento che verrebbero forniti a riga di comando uno ad uno nel caso di utilizzo di comando esteso.

Nella stesura del file di dataset, un parametro particolare è `enable_bucket`. Se configurato su `true`, il framework attiva una pipeline di bucketing dinamico che raggruppa le immagini in base a rapporti d'aspetto (*aspect ratio*) omogenei prima del campionamento latente, senza operare con un crop automatico a tutte le immagini. Tale approccio preserva le proporzioni native delle immagini durante l'addestramento, estendendo la flessibilità del modulo LoRA risultante anche in fase di inferenza su risoluzioni arbitrarie. Al contrario, impostandolo a `false` vincola la rete ad attendere matrici bidimensionali standardizzate e rigorosamente simmetriche.

Sebbene la modularità dei file TOML garantisca una scalabilità ideale, per operare con maggiore controllo sui vari parametri durante il lancio dei vari addestramenti e per effettuare un debugging più immediato, si è scelto di operare prevalentemente mediante l’invocazione del comando esteso direttamente da riga di comando.

5.1.2 Allocazione Hardware Distribuita in Modalità Multi-GPU

Una volta validata la sintassi di lancio in modalità a nodo singolo, la scalabilità dell’addestramento è stata estesa sfruttando l’architettura distribuita della partizione Booster di Leonardo. Il workflow è stato ingegnerizzato per parallelizzare il calcolo su tutte e 4 le GPU NVIDIA A100 allocate sul nodo di calcolo, adottando il paradigma del *Distributed Data Parallel* (DDP) [20].

Sotto il profilo analitico, la distribuzione del carico su 4 acceleratori grafici opera una divisione computazionale del tempo complessivo di training, frammentando il calcolo delle epoche in base al numero di dispositivi attivi. In questa configurazione, il *batch size* globale viene ripartito simmetricamente tra le GPU, riducendo drasticamente il tempo richiesto per l’elaborazione di ogni singolo ciclo di *backpropagation*.

Tuttavia, l’implementazione del calcolo Multi-GPU richiede lo scambio continuo di tensori di aggiornamento tra le schede grafiche. Questo meccanismo di sincronizzazione introduce un overhead di memoria statica e richiede un consumo addizionale di memoria video per configurare i buffer di comunicazione tra processi, mostrando come il vantaggio sia in termini di tempo e non di risparmio di VRAM. A prescindere dall’approccio Multi-GPU è stato necessario applicare delle tecniche di ottimizzazione e quantizzazione dei pesi per Flux.2, in modo da evitare il superamento dei limiti fisici dell’hardware (64 GB di VRAM).

5.2 Analisi e Ottimizzazione dei Training

Questa sezione analizza in chiave comparativa ed empirica l'addestramento dei moduli LoRA sui tre modelli generativi oggetto della sperimentazione, evidenziando le barriere architetturali incontrate e le relative soluzioni ingegneristiche specifiche applicate offline.

5.2.1 Flux.2

La complessità intrinseca del modello e il volume dei suoi parametri hanno costituito lo scoglio computazionale più critico dell'intera attività sperimentale, imponendo una radicale strategia di contenimento delle risorse.

A causa della totale assenza di connettività internet sui nodi di calcolo di Leonardo, l'invocazione dello script nativo di addestramento `train_flux_2_network.py` (all'interno del framework `musubi-tuner`) ha inizialmente riscontrato fallimenti sistematici dovuti all'impossibilità di interrogare i server remoti di Hugging Face per il download automatico dei tokenizer. Per aggirare tale blocco strutturale, si è proceduto alla replica manuale dell'esatta struttura ad albero del file system attesa dagli script. All'interno della directory radice di `musubi-tuner`, sono state iniettate le sottocartelle locali:

- `/openai/clip-vit-large/patch14`
- `/mistralai/Mistral-Small-3.1-24B-Instruct-2503`

In parallelo, è stato necessario effettuare un intervento di patching direttamente sul codice sorgente dello script Python `flux_utils.py`. Gli interventi si sono resi necessari per risolvere blocchi critici dovuti alla configurazione offline del supercomputer e a incompatibilità nell'architettura di caricamento del modello:

1. **Bypass del modulo Multimodale (da `AutoProcessor` ad `AutoTokenizer`):** originariamente, l'architettura di `Musubi Tuner` istanziava la classe estesa del `Processor` per caricare `Mistral`

3. In ambiente HPC offline, il tentativo di richiamare le funzionalità multimodali globali del processore ha generato eccezioni bloccanti (`OSError` e `ValueError`), causate dall'impossibilità di reperire o validare da remoto i file di configurazione visuale (come `preprocessor_config.json`). Per aggirare il problema senza stravolgere l'inizializzazione del modello, il metodo di *forward* è stato reingegnerizzato introducendo un controllo condizionale tramite la funzione `hasattr`. Se l'oggetto istanziato si configura come un `Processor`, lo script ne estrae dinamicamente la sola sotto-componente `.tokenizer`, in caso contrario, utilizza l'oggetto nativo. Questo patching ha permesso di isolare la pipeline puramente testuale a runtime, evitando le dipendenze multimodali superflue per il condizionamento di Flux.2.

2. Semplificazione della tokenizzazione e rimozione dei Chat

Template: è stata rimossa la chiamata alla funzione `apply_chat_template`, la quale formattava i prompt come una conversazione fittizia generando errori di validazione nel caricamento dei file di configurazione locali. Al suo posto, è stata introdotta un'elaborazione diretta del testo tramite il tokenizer puro.

3. Fissaggio della Lunghezza di Contesto (Forzatura a 256 to-

ken): a causa di un disallineamento software nell'ereditarietà delle variabili d'istanza (che generava un errore di tipo `AttributeError` sulla proprietà `max_length`), il parametro di troncamento e padding è stato impostato rigidamente a 256 token. Questa configurazione rispecchia lo standard nativo di condizionamento testuale richiesto dall'architettura di Flux.2, garantendo la corretta dimensionalità dei tensori generati e sbloccando l'esecuzione del data loader, permettendo così di procedere con l'addestramento.

Prima di avviare il training, è necessaria una procedura di pre-elaborazione del dataset strutturata in due fasi distinte e sequenziali, utile anche ad

alleggerire il carico computazionale durante le epoche di addestramento effettivo:

- `flux_2_cache_latents.py`: esegue il passaggio preliminare delle immagini d'addestramento attraverso il codificatore del Variational Autoencoder (VAE), estraendo le caratteristiche geometriche e memorizzandole in vettori latenti statici.
- `flux_2_cache_text_encoder_outputs.py`: elabora le stringhe di testo generate da Florence-2 passandole attraverso i Text Encoder (CLIP e Mistral), congelando i relativi embedding testuali.

I risultati di queste operazioni vengono salvati direttamente in file compatti in formato `.safetensors`. In questo modo, durante l'esecuzione del training, il modello non deve computare ripetutamente la codifica di immagini e testi a ogni epoca, azzerando i tempi di latenza legati all'I/O e riducendo il dispendio energetico complessivo.

```
accelerate launch --num_cpu_threads_per_process 1 --mixed_precision bf16 src/musubi_tuner/flux_2_train_network.py \
--model_version dev \
--dit path/to/dit_model \
--vae path/to/ae_model \
--text_encoder path/to/text_encoder \
--dataset_config path/to/toml \
--sdpa --mixed_precision bf16 \
--timestep_sampling flux2_shift --weighting_scheme none \
--optimizer_type adamw8bit --learning_rate 1e-4 --gradient_checkpointing \
--max_data_loader_n_workers 2 --persistent_data_loader_workers \
--network_module networks.lora_flux_2 --network_dim 32 \
--max_train_epochs 16 --save_every_n_epochs 1 --seed 42 \
--output_dir path/to/output_dir --output_name name-of-lora
```

Figura 5.2: Comando di Addestramento Flux.2

È stato attivato il meccanismo di *gradient checkpointing* che è progettato per liberare la memoria eliminando le attivazioni dei layer intermedi durante il calcolo in avanti per poi ricalcolarle nel calcolo inverso. Nonostante questo, l'allocazione standard di Flux.2 ha generato sistematici errori di saturazione della memoria video. L'analisi analitica dell'impatto statico dei pesi evidenzia la severità del problema.

Il DiT (Transformer) ha 12 miliardi di parametri e occupa 24 giga, il text encoder T5-XXL occupa 22 giga e clip e vae ne occupano altri 2. Quindi, il semplice caricamento statico dei pesi del modello nello

spazio della memoria video richiede circa 48 GB di VRAM. I restanti 16 GB disponibili sui 64 GB fisici della GPU NVIDIA A100 si sono rivelati matematicamente insufficienti a ospitare i tensori intermedi delle attivazioni e l'overhead introdotto dalla comunicazione Multi-GPU. Per stabilizzare il job prima dell'avvio della prima epoca, si è reso indispensabile dimezzare la precisione, utilizzando i flag `fp8_base`, `fp8_scaled` e `fp8_tokenizer` nel comando esteso. La compressione matematica delle matrici a 8 bit ha consentito di dimezzare l'occupazione statica dei pesi, riducendo l'impatto a runtime e permettendo il corretto avvio dell'addestramento.

Validata la stabilità dell'ambiente, sono stati eseguiti diversi addestramenti approfonditi di modelli LoRA sui vari soggetti menzionati nei capitoli precedenti. Prendendo in esame la configurazione di training per “Ati” analizzata anche per SD 1.5 nel Capitolo 2 su un dataset di 70 immagini, sono stati impostati i seguenti parametri:

- **Rank:** 64.
- **Alpha:** 32.
- **Learning Rate:** 5×10^{-5} , valore ridotto del 50% rispetto ai benchmark standard applicati su Stable Diffusion 1.5 per prevenire fenomeni di divergenza del gradiente su Transformer.
- **Ripetizioni:** 4.
- **Epoche:** 8.

L'esecuzione ha registrato un tempo di calcolo complessivo di circa 50 minuti per il completamento dei 2240 step totali. Grazie alle ottimizzazioni applicate, la VRAM si è attestata su un picco iniziale di 50 GB durante la fase critica di allocazione, per poi stabilizzarsi sui 47 GB durante la discesa del gradiente. Una volta terminato il training alcuni dei file delle epoche sono stati utilizzati all'interno di appositi workflow di ComfyUI, confermando l'onerosità dell'ecosistema Flux.2. Infatti, il

semplice avvio del workflow di test all'interno dell'ambiente ComfyUI ha richiesto l'allocazione statica di ben 60 GB di VRAM.

Tabella 5.1: Andamento della Loss per Epoca nell'addestramento di Flux.2.

Epoca	Steps	Average Loss
1	70	0.574
2	140	0.566
3	210	0.553
4	280	0.551
5	350	0.537
6	420	0.525
7	490	0.518
8	560	0.533

La funzione di perdita ha manifestato una traiettoria discendente e priva di fluttuazioni anomale, riducendosi in modo controllato senza mai scendere al di sotto del valore di guardia di 0.50. Questa dinamica dimostra che sono stati scongiurati fenomeni latenti di overfitting o memorizzazione meccanica dei pixel del dataset. Il punto di minimo matematico è stato intercettato in corrispondenza dell'Epoca 7 (Loss = 0.518), mentre l'Epoca 8 ha registrato una risalita infinitesimale (Loss = 0.533, con uno scarto trascurabile di appena 0.015). Una variazione così esigua implica che lo scostamento numerico non definisce univocamente la qualità formale dell'output. L'inversione della curva della loss identifica tuttavia nell'Epoca 7 il punto di equilibrio ottimale tra l'ancoraggio semantico dei tratti del reperto e la stabilità morfologica del modello. Come si può notare in Figura 5.3, vengono mostrati dei risultati di generazione che dimostrano l'ottima qualità dei risultati finali.



Figura 5.3: Risultati di generazione per il soggetto Ati con modulo LoRA Flux.2.

5.2.2 Flux.1

L'analisi degli addestramenti svolti su Flux.1 offre un perfetto termine di paragone per quantificare i vantaggi di un'architettura meno pesante in termini di allocazione delle risorse e permette un paragone diretto con Flux.2.

L'addestramento di Flux.1 è stato condotto sfruttando l'architettura classica di `sd-scripts`. Per superare l'isolamento della rete offline sui nodi di calcolo di Leonardo, è stata replicata una struttura speculare a quella descritta in precedenza per Flux.2, iniettando manualmente nel file system del cluster le directory locali:

- `/openai/clip-vit-large/patch14`
- `/google/t5-v1_1-xxl/`

Una differenza architetturale sostanziale rispetto a musubi-tuner risiede nella gestione del pre-processing, infatti lo script `train_flux_network.py` di Kohya esegue la fase di caching dei vettori latenti e degli

embedding testuali in modalità completamente automatizzata all'avvio del job. Il framework alloca temporaneamente le risorse per calcolare e congelare i tensori grafici e linguistici senza richiedere l'invocazione manuale di script separati di pre-elaborazione, ottimizzando i tempi di setup operativi.

```
accelerate launch --num_cpu_threads_per_process 1 flux_train_network.py \
--pretrained_model_name_or_path="/leonardo_scratch/large/userexternal/mmancin2/kohya_install/models/flux1/transformer/flux1-dev.safetensors" \
--clip_l="openai/clip-vit-large-patch16/model.safetensors" \
--tokenizer1="google/gpt2-v1.1-xxl/xxl-1p16.safetensors" \
--vae="/leonardo_scratch/large/userexternal/mmancin2/kohya_install/models/flux1/vae/ae.safetensors" \
--dataset_config="dataset_flux1_ati.toml" \
--output_dir="/leonardo_scratch/large/userexternal/mmancin2/kohya_install/outputs/flux1/ati_v1" \
--output_name="Ati_flux1_v1" \
--save_model_as_safetensors \
--network_module=networks.lora_flux \
--network_dim=64 \
--network_alpha=32 \
--learning_rate=5e-5 \
--optimizer_type="Adam8Bit" \
--lr_scheduler="constant" \
--sdp \
--max_train_epochs=10 \
--save_every_n_epochs=1 \
--mixed_precision="bf16" \
--gradient_checkpointing \
--guidance_scale=1.0 \
--timestep_sampling="flux_shift" \
--model_prediction_type="raw" \
--blocks_to_swap=19 \
--cache_text_encoder_outputs \
--cache_latents \
--persistent_data_loader_workers \
--max_data_loader_n_workers 2 \
--disable_mmap_load_safetensors
```

Figura 5.4: Comando di Addestramento Flux.1

Sotto il profilo della gestione della memoria video, Flux.1 è caratterizzato da un consumo di memoria nettamente inferiori a runtime, questo ha permesso di completare l'addestramento senza la necessità di ricorrere a ottimizzazioni di quantizzazione a 8 bit, operando in precisione nativa a virgola mobile. Il monitoraggio della VRAM ha registrato un picco massimo di occupazione pari a soli 23 GB, un valore ampiamente al di sotto della soglia critica dei 64 GB.

Valutando l'addestramento svolto su “Ati” (70 immagini), sono stati impostati i seguenti parametri:

- **Rank:** 64
- **Alpha:** 32
- **Learning Rate:** 5×10^{-5}
- **Ripetizioni:** 4
- **Epoche:** 8

Tabella 5.2: Andamento della Loss per Epoca nell'addestramento di Flux.1.

Epoca	Steps	Average Loss
1	70	0.345
2	140	0.347
3	210	0.331
4	280	0.327
5	350	0.325
6	420	0.338
7	490	0.317
8	560	0.322

Il sistema ha mantenuto una reattività costante pari a circa 1.04 s/it, portando a compimento i 2240 step totali in un tempo di calcolo effettivo di 38 minuti e 54 secondi. Il valore di *average loss* ha mostrato un trend decrescente e stabile, partendo da 0.345 al termine della prima epoca e raggiungendo un valore finale di 0.322. Questa stabilità indica una buona convergenza del modello senza evidenti fenomeni di overfitting o instabilità numerica durante le fasi di backpropagation.



Figura 5.5: Risultati di generazione per il soggetto Ati con modulo LoRA Flux.1.

5.2.3 Z-Image

La sperimentazione condotta su Z-Image ha risposto alla necessità di validare un'architettura generativa radicalmente differente rispetto alla

famiglia Flux, orientando la ricerca verso modelli ottimizzati per l'agilità computazionale senza compromessi sulla fedeltà semantica. Nello specifico, si è scelto di adottare la versione Z-Image Base anziché quella Turbo vista nel caso di generazione su hardware consumer. Sebbene la versione Turbo prediliga l'estrema velocità di campionamento riducendo drasticamente il numero di step di inferenza, la versione Base conserva una superiore potenza espressiva e una maggiore precisione nel riflettere i vincoli filologici dei prompt, configurandosi come la soluzione ideale per un addestramento scientifico ad alta fedeltà su infrastrutture HPC.

L'addestramento utilizza lo script `zimage_train_network.py` integrato nel framework `musubi-tuner`. Come per Flux.2, è stata eseguita la configurazione del file system offline, iniettando i tokenizer locali all'interno della radice del modulo ed evitando le chiamate remote ai server Hugging Face.

```

/ ../musubi-tuner/z-image-base/
├─ dit/
│   ├── 00001-of-00002.safetensors
│   └── 00002-of-00002.safetensors
├─ text_encoder/
│   └── model.safetensors (Qwen)
└─ vae/
    ├── 00001-of-00002.safetensors
    └── 00002-of-00002.safetensors
tokenizer/
├─ tokenizer_00001-of-00003.safetensors
├─ tokenizer_00002-of-00003.safetensors
├─ tokenizer_00003-of-00003.safetensors
└─ vae/
    └─ ae.safetensors

```

Figura 5.6: Struttura delle cartelle locali per l'addestramento offline di Z-Image.

Il modello poggia sulla pipeline di caching preventivo a due fasi vista per Flux.2. Gli script `zimage_cache_latents.py` e `zimage_cache_text_encoder_outputs.py` estraggono preventivamente le caratteristiche delle immagini e memorizzano le informazioni dei file testuali.

```

accelerate launch --num_cpu_threads_per_process 1 --mixed_precision bf16 src/musubi_tuner/zimage_train_network.py \
--dit 7/leonardo_scratch/large/userexternal/mancin2/kohya_install/large/musubi_tuner/tongyi-Mat2-Image/transformer/diffusion_pytorch_model-00001-of-00002.safetensors" \
--vae 7/leonardo_scratch/large/userexternal/mancin2/kohya_install/musubi_tuner/tongyi-Mat2-Image/vae/diffusion_pytorch_model.safetensors" \
--text_encoder 7/leonardo_scratch/large/userexternal/mancin2/kohya_install/musubi_tuner/tongyi-Mat2-Image/text_encoder/model-00001-of-00003.safetensors" \
--dataset_config "dataset_2_Ati1.toml" \
--vqga --mixed_precision bf16 \
--timestep_sampling shift --weighting_scheme none --discrete_flow_shift 2.0 \
--optimizer_type adamw_torch --learning_rate 1e-4 \
--gradient_checkpointing \
--max_data_loader_n_workers 2 --persistent_data_loader_workers \
--network_config networks.lora_zimage \
--network_dim 64 \
--network_alpha 32 \
--max_train_epochs 12 \
--save_every_n_epochs 1 \
--seed 42 \
--output_dir 7/leonardo_scratch/large/userexternal/mancin2/kohya_install/outputs/zimage/zati_v1" \
--output_name "Ati_2"

```

Figura 5.7: Comando di Addestramento Z-Image

Considerando sempre Ati come soggetto e il dataset composto da 70 immagini, i valori dei parametri sono:

- **Rank:** 64
- **Alpha:** 32
- **Learning Rate:** 5×10^{-5}
- **Ripetizioni:** 4
- **Epoche:** 12

Questa configurazione porta ad un totale di 6720 step distribuiti stabilmente sulle 4 GPU del nodo Booster, completando il training in 50 minuti. L'occupazione della VRAM si è stabilizzata sui 20 GB, operando in precisione nativa ed eliminando all'origine la necessità di flag di quantizzazione.

Tabella 5.3: Andamento della Loss per Epoca nell’addestramento di Z-Image.

Epoca	Steps	Average Loss
1	140	0.368
2	280	0.375
3	420	0.381
4	560	0.358
5	700	0.353
6	840	0.351
7	980	0.353
8	1120	0.382
9	1260	0.349
10	1400	0.333
11	1540	0.349
12	1680	0.355

Sotto il profilo della convergenza, la curva della Loss Function ha esibito un comportamento dinamico ma solido, privo di picchi di instabilità numerica, raggiungendo un valore minimo di perdita pari a 0.333 all’epoca 10. Questo punto di minimo ha identificato il checkpoint ottimale del modulo LoRA, intercettando il perfetto equilibrio tra la memorizzazione dei dettagli morfologici dei reperti etruschi e la capacità di generalizzazione stilistica nello spazio latente.

In fase di generazione e testing dei modelli addestrati si è notata un’importante differenza rispetto ai modelli Flux. La generazione di Ati è risultata leggermente più imprecisa e prona ad errori rispetto agli altri due modelli. Invece le generazioni degli altri soggetti hanno mantenuto ottimi standard di qualità. Nella Figura 5.8 vengono mostrati dei risultati dei vari soggetti utilizzati.



Figura 5.8: Risultati di generazione dei soggetti estratti con Z-Image Base.

5.2.4 Sintesi Comparativa dei Processi di Addestramento

È possibile tracciare un quadro comparativo che metta in relazione le performance computazionali, i vincoli hardware e le dinamiche di convergenza registrate sull'infrastruttura HPC Leonardo. La strutturazione di questa sintesi permette di quantificare l'efficienza ingegneristica delle diverse architetture a parità di dataset e parametri di base.

Nella Tabella 5.4 viene schematizzato il comportamento dei tre modelli, evidenziando il netto contrasto tra l'onerosità computazionale della famiglia Flux e l'agilità strutturale di Z-Image.

Tabella 5.4: Quadro comparativo delle metriche prestazionali, software e hardware su Leonardo HPC.

Parametro Metrico e Strutturale	Flux.2 Dev	Flux.1	Z-Image
Architettura del Core	MMDiT (~32B parametri)	DiT (~12B parametri)	S3-DiT (~6B parametri)
Framework Software	musubi-tuner	sd-scripts	musubi-tuner
Gestione del Caching	Manuale a due fasi	Automatizzata integrata	Manuale a due fasi
Precisione ed Ottimizzazione	Quantizzazione FP8	Precisione Nativa (FP16/BF16)	Precisione Nativa (FP16/BF16)
Occupazione VRAM (Plateau)	~47 - 50 GB	~23 GB	~20 GB
Velocità di Elaborazione	~1.34 s/it	~1.04 s/it	~0.44 s/it
Configurazione di Target	2240 step (4 epoche)	2240 step (8 epoche)	6720 step (12 epoche)
Stabilità della Loss Function	Discendente e stabilizzata	Discendente e stabilizzata	Discendente e stabilizzata

Dall'analisi si nota come Flux.2 integra un numero di parametri globali molto elevato rispetto agli altri due modelli, data l'architettura a flusso congiunto che raddoppia l'impatto delle attivazioni a runtime. Questo rende la quantizzazione FP8 l'unico canale abilitante per evitare il collasso della VRAM.

Z-Image dimostra una scalabilità ottimale legata alla sua snellezza architetturale. Il dimezzamento dei parametri del core e l'introduzione di connessioni a corto raggio traslate non riducono l'allocazione temporale complessiva del job, ma permettono di elaborare una frequenza di calcolo nettamente superiore. Ciò consente al modello di raggiungere il proprio target naturale di convergenza a 6720 step nello stesso intervallo in cui le architetture Flux completano i propri 2240 step di target.

Sotto il profilo statistico, tutte e tre le funzioni di perdita hanno manifestato un comportamento analogo e altamente stabile, caratterizzato da una discesa controllata nelle prime epoche e dal successivo raggiungimento di una stabilizzazione.

5.3 Prompt Engineering e Ingegnerizzazione Sintattica in Ambienti DiT

Mentre le vecchie architetture rispondevano in modo ottimale a stringhe di testo frammentate, composte da elenchi di parole chiave isolate e separati da virgole (*tag-based prompting*), i modelli di nuova generazione prediligono una sintassi discorsiva e fluida.

In questo paragrafo viene mostrata l’ingegnerizzazione sintattica applicata alle generazioni di Ati, evidenziando la capacità dei diversi modelli di decodificare e iniettare nel pixel space i dettagli protetti dal modulo LoRA.

5.3.1 Strutturazione Gerarchica del Prompt e Analisi della Resa Materica

Nelle simulazioni standard volte a riprodurre il soggetto immerso in un contesto naturale coerente con l’ambiente archeologico, la sintassi del prompt è stata strutturata secondo una rigida gerarchia semantica. Un esempio di prompt è il seguente:

“AtiFlux2, an half body image of a 3d model woman in an orange dress with an orange hat. The orange dress has a texture with little crosses with a nature background. She’s smiling and looking to the left. Upper body image.”



Figura 5.9: Output di Ati ottenuto con il prompt citato sopra utilizzando Flux.2

La struttura del prompt è composta da quattro blocchi funzionali:

1. **Token di Invocazione:** parola chiave univoca associata al modulo LoRA durante la fase di training su Leonardo. Questo elemento agisce come attivatore tensoriale, direzionando l'inferenza verso i pesi specifici del modulo inserito.
2. **Definizione della Posa e Inquadratura:** vincoli geometrici espliciti impostano i piani focali dell'inquadratura virtuale.
3. **Descrizione Morfologica e Dettaglio delle Texture:** specifica le caratteristiche dell'abbigliamento del soggetto.
4. **Condizionamento Ambientale ed Espressivo:** determina l'interazione con lo sfondo e i parametri dell'espressione facciale.

Sono stati evidenziati divari architetturali tra i modelli in esame. Nelle generazioni condotte con Flux.1 e Z-Image, l'esplicitazione della trama del vestito ha riscontrato severe barriere di aderenza semantica. I due

modelli hanno manifestato fenomeni di saturazione cromatica ed approssimazione geometrica, collassando i dettagli dei pattern in una campitura omogenea di colore rosso-arancione.

Al contrario, l'inferenza guidata da Flux.2 ha esibito un livello di risoluzione micro-strutturale notevolmente superiore: l'architettura MM-DiT è stata in grado di mappare correttamente la texture geometrica a crocette sulla superficie tridimensionale dell'abito, isolando i canali di informazione e garantendo la fedeltà filologica del reperto.

5.3.2 Test di Generalizzazione Spaziale e Iniezione di Stili Esogeni

Per saggiare la plasticità dello spazio latente e valutare la capacità del modulo LoRA di svincolare il soggetto dall'ambiente storico presente nel dataset di addestramento, sono stati condotti test di generalizzazione iniettando nel prompt contesti ambientali ed elementi grafici estranei agli sfondi utilizzati nel dataset.

Un caso di studio particolarmente significativo è rappresentato dalla ricollocazione del soggetto in un ambiente marino, associata alla richiesta di generazione di una scritta. Un esempio di prompt utilizzato per questo tipo di generazione è:

“AtiFlux2, a 3d model of a woman in an orange dress with an orange hat and brown shoes. The orange dress has a texture with little crosses. Beach and sea background with a tree on the left.”



Figura 5.10: Ati in un background marittimo

I risultati visivi offrono importanti considerazioni:

- **Disaccoppiamento semantico:** Flux.2 dimostra una capacità ottimale di *disentanglement* (disaccoppiamento). Il modello isola accuratamente i pesi legati alla geometria tridimensionale di Ati e alla texture del vestito, senza generare allucinazioni all'interno del nuovo panorama costiero.
- **Resa del testo nativo e vincoli grafici:** l'architettura risponde con estrema precisione alle relazioni spaziali e logiche descritte nel prompt, posizionando i nuovi elementi dove richiesto.

5.4 Tecniche di Inpainting

Una volta consolidati i checkpoint ottimali per i singoli soggetti, la ricerca si è orientata verso la validazione qualitativa e funzionale dei moduli generati. Questa fase ha esplorato i limiti di tolleranza delle reti di fronte a sollecitazioni complesse, quali l'interazione simultanea di più Lo-

RA, il ripristino localizzato di porzioni d'immagine e il condizionamento geometrico spaziale.

Per ovviare alle imperfezioni intrinseche della generazione diretta e inserire elementi integrativi preservando l'esattezza storica dei reperti, sono state testate e confrontate due differenti metodologie di Inpainting all'interno dell'ecosistema ComfyUI.

5.4.1 Inpainting tramite Mask Editor

Il primo approccio è quello di effettuare inpainting direttamente sull'immagine di input creando una maschera grazie alla funzionalità Mask Editor messa a disposizione da ComfyUI. Questo approccio opera una segmentazione geometrica rigida sulla matrice dei pixel dell'immagine sorgente. L'operatore definisce manualmente una maschera direttamente sull'immagine di input nel workflow ComfyUI. Il modello campiona esclusivamente l'area latente corrispondente alla maschera, eseguendo un processo di denoising condizionato dal nuovo prompt testuale, mentre i pixel esterni vengono congelati.

- **Vantaggi:** garantisce un controllo spaziale assoluto e rigoroso sulla delimitazione del nuovo elemento archeologico.
- **Criticità:** l'Inpainting locale risente fortemente dell'estensione dell'area mascherata. Se la superficie è geometricamente troppo esigua, la rete non dispone dello spazio latente necessario per sviluppare la complessità anatomica o strutturale richiesta dal prompt. Inoltre, la discontinuità tra la zona in addizione e lo sfondo statico genera frequentemente artefatti visivi in corrispondenza del perimetro della maschera.

Per utilizzare le maschere all'interno del workflow di ComfyUI, il nodo VAE Encode ha il parametro "mask", in questo modo vede la maschera. Quindi, nella fase di denoising il modello opera solo nella zona definita manualmente dall'operatore. Il parametro "grow_mask_by" permette di espandere l'area della maschera.

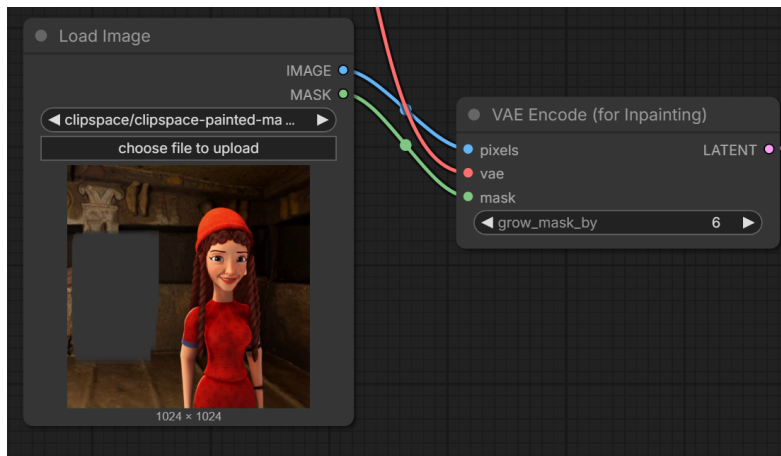


Figura 5.11: Nodo per maschera in ComfyUI

Un esempio di utilizzo di questa tecnica è mostrato in 5.12, in cui è stata prima generata l'immagine realistica, poi creata la maschera manualmente e inserita l'antefissa grazie all'utilizzo del Relativo LoRA.



Figura 5.12: Inpainting con maschera manuale

5.4.2 Inpainting tramite Workflow apposito

Il secondo approccio è basato su un template nativo di Flux.2 Dev in ComfyUI, sfruttando una tecnica alternativa che elimina il concetto di mascheramento manuale. Il workflow prevede la rigenerazione completa dell'immagine partendo da zero, utilizzando però l'immagine iniziale come mappa di guida globale per la generazione di quella finale. Nel

prompt è sufficiente descrivere brevemente l'immagine e l'aggiunta che si intende inserire.

- **Vantaggi:** la fusione stilistica e l'integrazione della luce risultano impeccabili e prive di artefatti perimetrali. Il modello ricostruisce l'intera scena armonizzando il nuovo soggetto con lo stile dell'immagine di input e degli elementi già esistenti.
- **Criticità:** c'è una parziale perdita di controllo. Dato che la rete esegue il denoising sull'intero piano dell'immagine, può accadere anche se raramente che piccoli dettagli di sfondo non direttamente mappati nel prompt protetto possono subire lievi mutazioni morfologiche indesiderate.

In figura 5.13 si può osservare il workflow che permette di effettuare inpainting in questo modo. Data l'immagine di input, viene fatto un rapido upscale (opzionale) e grazie al nodo Get Image Size vengono impostate le dimensioni dell'immagine di output in maniera automatica. Gli altri elementi sono quelli già visti e comuni in molti workflow, a parte per la parte relativa al KSampler che è stato frammentato in vari componenti in modo da avere ancora più controllo

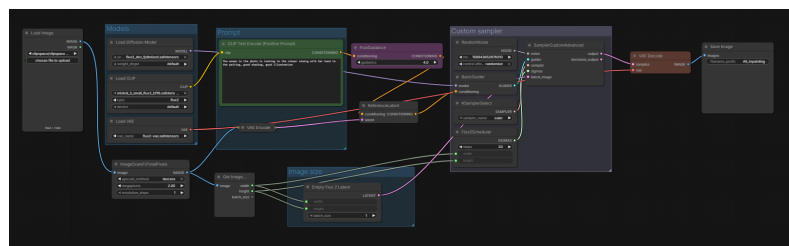


Figura 5.13: Workflow ComfyUI - Inpainting Flux.2

Per generare l'immagine visibile nella Figura 5.14 è stato utilizzato questo prompt:

“On the right side of the image there is an orange cat”



Figura 5.14: Inpainting con il secondo metodo

Uno scenario d'uso molto interessante è stato quello di partire da una scena vuota e inserire il soggetto grazie all'utilizzo del LoRA. Questo metodo è stato utilizzato soprattutto nella fase di generazione video che verrà mostrata nel Capitolo 5, per modificare l'immagine di partenza.

5.5 Integrazione Simultanea di Multi-LoRA

La generazione di scene in cui convivono sia Ati all'interno che il Tempio Etrusco o l'Antefissa richiede la sovrapposizione in sequenza di matrici LoRA distinte all'interno del medesimo grafo generativo di ComfyUI. Tale operazione si scontra con il fenomeno dell'interferenza dei pesi (*Weight Blending*). L'iniezione simultanea di vettori di aggiornamento esterni sulle medesime matrici di proiezione dei blocchi di attenzione può causare distorsioni semantiche o allucinazioni stilistiche.

I test combinatori eseguiti hanno evidenziato risposte radicalmente divergenti da parte dei tre modelli:

- **Flux.2:** ha esibito una capacità di isolamento concettuale nettamente superiore. L'imponente dimensionalità del suo spazio latente permette di mappare il soggetto in primo piano e l'architettura di sfondo su canali ortogonali, riducendo al minimo la contaminazione incrociata (*cross-contamination*).

- **Flux.1 e Z-Image:** hanno mostrato una maggiore sensibilità al merge dei pesi. In assenza di una calibrazione millimetrica, l'introduzione dell'antefissa tende a corrompere i dettagli morfologici di Ati, trasferendo la texture dell'oggetto direttamente sulla geometria del personaggio.

Sotto il profilo analitico, la mitigazione di tali artefatti è stata ottenuta definendo empiricamente una finestra di bilanciamento matematico dei pesi LoRA. È stato necessario impostare il modulo del soggetto principale a una forza dominante, riducendo linearmente l'apporto del modulo di sfondo o del reperto:

$$W_{\text{Ati}} = 0.80 \dots 0.85$$

$$W_{\text{Sfondo/Reperto}} = 0.70 \dots 0.75$$

Scendere sotto la soglia di 0.75 per il soggetto Ati determina un immediato decadimento della fedeltà filologica dei tratti somatici, mentre il superamento del valore di 0.80 sul secondo LoRA satura lo spazio latente, provocando il collasso dei dettagli dello sfondo.

5.5.1 Flux.2

È stata affrontata la stabilità architettonica in scenari multi-LoRA per Flux.2. Il workflow finale che ha portato alla generazione di immagini combinando i vari LoRA è mostrato in figura 5.15.

La forza dei modelli LoRA è facilmente impostabile grazie al parametro “strength model”. La particolarità del workflow di Flux.2 è il nodo **FluxGuidance** che bilancia dinamicamente la creatività del modello e la fedeltà al testo, riducendo al minimo gli errori dei modelli precedenti.

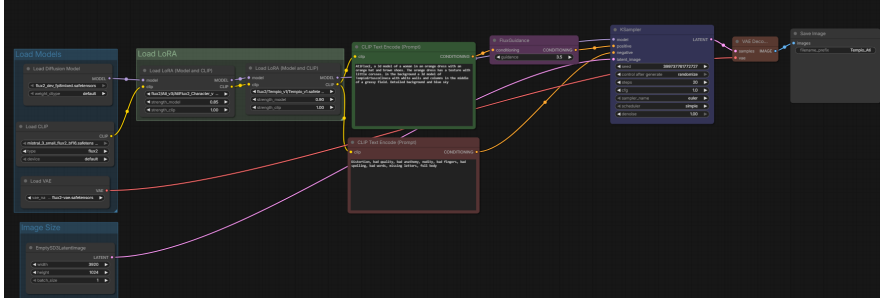


Figura 5.15: Workflow Flux.2

Le immagini mostrate in figura 5.16 sono state generate utilizzando entrambi i nodi di caricamento dei modelli LoRA in sequenza e il prompt:

“AtiFlux2, a 3d model of a woman in an orange dress with an orange hat and brown shoes. She’s in a museum presenting a 3d model image Antefissa1Cineca of a stone with a face on the center in a glass display case”

Dall’analisi comparativa degli output emerge una spiccata asimmetria generativa in base al bilanciamento dei pesi. Infatti, nella porzione sinistra della figura si rileva una marcata perdita di definizione geometrica dell’antefissa, associata a un fenomeno di bleeding stilistico che corrompe i pattern volumetrici del vestito del soggetto. Al contrario, la porzione destra evidenzia un punto di equilibrio più controllato, sebbene permanga una quota minimale di weight merge inter-concettuale. Tale evidenza empirica conferma l’assenza di un principio di convergenza universale, riaffermando la centralità delle procedure di *trial and error* nella *fine-tuning* dei parametri operativi.



Figura 5.16: Analisi dell'interferenza semantica nella combinazione dei moduli LoRA (Ati + Antefissa) in Flux.2.

Le sessioni sperimentali indicano che una riduzione del peso associato ad Ati sotto la soglia critica di 0.75 compromette severamente la fedeltà morfologica dei tratti identitari del personaggio. Il punto di bilanciamento è stato identificato impostando il fattore di scala dominante per il soggetto principale a $W_{\text{Ati}} = 0.80$, abbinato a una modulazione del secondo LoRA compresa tra 0.70 e 0.75. Questo intervallo garantisce una corretta varianza per gli elementi di sfondo senza saturare lo spazio latente del modello. Valori simili sono stati utilizzati anche per la combinazione con gli altri soggetti, ovvero la Tomba Etrusca e il Tempio Etrusco.



Figura 5.17: Output generativi risultanti dalla combinazione sequenziale dei moduli LoRA per la Tomba Etrusca e il Tempio Etrusco in associazione al soggetto Ati (Flux.2).

5.5.2 Flux.1

Anche l'architettura Flux.1 è stata sottoposta a protocolli di stratificazione sequenziale per mappare la tolleranza del modello alla sovrapposizione di matrici di rango ridotto. La risposta del modello ha mostrato una dipendenza dai modelli iniziali. La combinazione simultanea di Ati con il modulo del Tempio ha prodotto configurazioni spaziali accettabili anche se non precise, mentre l'interazione con l'antefissa ha evidenziato forti criticità operative.

In figura 5.18 è possibile osservare il workflow finale utilizzato nel caso di Flux.1.

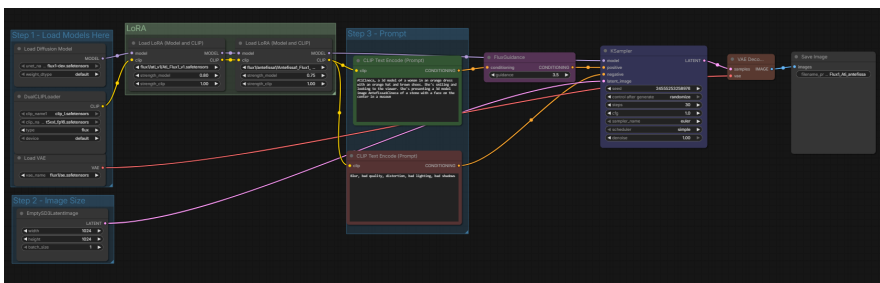


Figura 5.18: Workflow Flux.1

In termini comparativi, Flux.1 dimostra una resistenza all'interferenza semantica marcatamente inferiore rispetto a Flux.2. Questa debolezza strutturale si traduce in una ridotta efficienza generativa, rendendo necessaria una rigorosa selezione a posteriori (*cherry-picking*) degli output privi di aberrazioni su campionari estesi. Il modello tende a prendere ampi margini di libertà interpretativa in base al bilanciamento dei pesi, portando frequentemente ad avere allucinazioni o alla cancellazione dei tratti distintivi dei soggetti.

Nel caso altamente critico dell'antefissa, per scongiurare il totale collasso dei dettagli ad alta frequenza del reperto, si è resa necessaria una sovra-amplificazione del relativo modulo LoRA, accettando come inevitabile trade-off una parziale degradazione della fedeltà filologica del personaggio principale. Il prompt utilizzato in questo caso è:

“AtiCineca, a 3d model of a woman in an orange dress with an orange hat and brown shoes. She’s smiling and looking to the viewer. She’s presenting in a museum a 3d model image Antefissa1Cineca of a stone with a face on the center”



Figura 5.19: Effetti di sovrapposizione concettuale e allucinazioni formali in scenari multi-LoRA su architettura Flux.1.

5.5.3 Z-Image

Per quanto riguarda Z-Image, i test hanno mostrato le vulnerabilità alle interferenze stilistiche. In figura 5.20 viene mostrato il workflow finale utilizzato per la generazione con Z-Image. La particolarità in questo caso è il nodo **ModelSamplingAuraFlow**. Infatti, Z-Image è basato su un'architettura S3-DiT che deriva direttamente da AuraFlow. È un nodo utilizzato per configurare e ottimizzare la pianificazione dei passi di campionamento specificamente per modelli di generazione di immagini AuraFlow in modo da ottenere l'effetto desiderato. Il nodo ha un parametro, "shift", che influenza il comportamento dell'algoritmo di campionamento, incidendo sulle modalità con cui il modello passa da uno stato all'altro durante il processo di generazione. La regolazione di questo valore contribuisce a definire l'output del modello per ottenere l'effetto desiderato. Un esempio di prompt utilizzato in questo workflow è:

"A 3D model of a woman AtiZImage in an orange dress with an orange hat and braids, presenting a 3d model image of a Antefissa1Cineca stone with a face on the center. It is in a glass case on display in a museum. High quality background, photorealistic lighting"

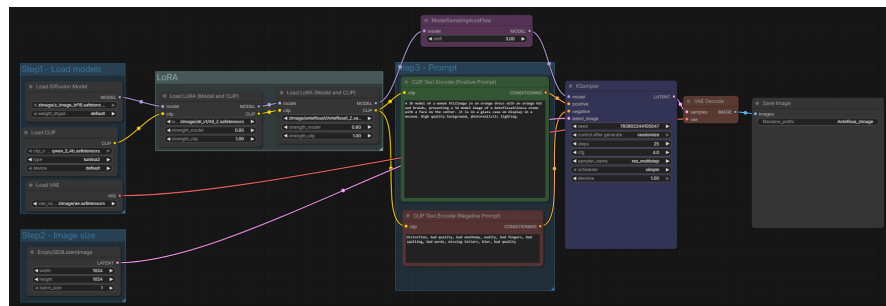


Figura 5.20: Workflow Z-Image.

Sebbene gli oggetti singoli vengano generati in maniera molto fedele nella maggior parte dei casi, Z-Image non ha portato buoni risultati nel momento in cui si vanno a combinare due LoRA. Infatti, come si può vedere dalla Figura 5.21, l'antefissa va ad influenzare molto lo stile di Ati, portando a risultati non accettabili.



Figura 5.21: Fenomeni di bleeding materico e degradazione stilistica nell'interazione multi-LoRA su Z-Image.

Un'altra criticità riscontrata con Z-Image è stata quella legata al controllo della geometria dell'immagine finale, in modo da sopperire alla degradazione geometrica e stilistica riscontrata dai test. Per svolgere questo compito è necessario utilizzare dei modelli specifici chiamati ControlNet.

Tuttavia, al momento della sperimentazione, l'architettura S3-DiT soffriva della totale assenza di moduli ControlNet integrati nativamente nella struttura di ComfyUI, a differenza dei consolidati strumenti disponibili per la famiglia Flux. Per superare tale limitazione e consentire il controllo della posa e della coerenza strutturale, si è reso necessario implementare nodi custom di terze parti e integrare il modello specializzato Z-Image-Fun-ControlNet Union.

Nonostante l'integrazione di queste soluzioni permetta un controllo maggiore sulla composizione, la natura dell'architettura S3-DiT di Z-Image sembra rendere il modello particolarmente sensibile alle interferenze stilistiche quando si operano trasformazioni complesse o si combi-

nano più LoRA, confermando la tendenza del modello a privilegiare la coerenza del singolo soggetto rispetto a composizioni stratificate.

5.6 Analisi Comparativa di Sampler e Scheduler: Coerenza Visiva e Stabilità Metrica

Sono state effettuate diverse comparazioni tra combinazioni sampler scheduler, in modo da comprendere tramite l’osservazione delle immagini generate la variazione e la precisione di ciascuna combinazione rispetto alle precedenti.

Questi test sono stati svolti per capire quale combinazione portasse ad un risultato più stabile tra una generazione e l’altra, in modo da sfruttare questa proprietà in caso di bisogno di una sequenza di generazioni filologicamente corrette e coerenti tra loro.

5.6.1 Metodologia di Testing e Risultati Empirici

I test sono stati condotti usando il modello Flux.2 Dev. Il protocollo sperimentale ha previsto l’esecuzione di due generazioni sequenziali per ciascuna combinazione, mantenendo rigidamente invariati tutti i parametri e andando a cambiare unicamente qualche piccola informazione nel prompt.

Il prompt iniziale ha definito un soggetto standardizzato neutro, un cane in movimento all’interno di un parco, caratterizzato da dettagli specifici:

“A Border Collie dog running in a park with a green tennis ball in his mouth, with a blue collar”

Nella seconda generazione, il prompt è stato alterato esclusivamente in corrispondenza di due attributi (colore della palla e del collare) e del vettore di illuminazione:

“A Border Collie dog running in a park with a yellow tennis ball in his mouth, with a red collar. Autumn lighting”

È stato valutato l'indice di conservazione anatomica della scena, osservando la morfologia della zampa posteriore destra, la tessitura del pelo e il posizionamento della coda, oltre allo stile generale dell'immagine.

I risultati qualitativi e di coerenza visiva registrati sono stati catalogati secondo tre classi metriche:

- **Positivo:** stabilità anatomica, totale assenza o minima variazione di artefatti geometrici, variazione esclusiva degli elementi target richiesti dal prompt.
- **Misto:** accettabilità qualitativa generale, ma accompagnata da piccole mutazioni stilistiche o lievi alterazioni strutturali nei dettagli non protetti, leggermente evidenti.
- **Negativo:** collasso completo dello spazio latente, immagini sature, distorte, irriconoscibili o totale incoerenza morfologica tra la prima e la seconda generazione.

I dati emersi dalla campagna di test sono sintetizzati nella Tabella 5.5.

Scheduler / Sampler	Euler	Heun	dpm_2	dpmpp_2m
Simple	Positivo	Positivo	Positivo	Positivo
Sgm_uniform	Positivo	Positivo	Positivo	Positivo
Karras	Negativo	Negativo	Negativo	Negativo
Exponential	Negativo	Negativo	Negativo	Negativo
Ddim_uniform	Positivo	Positivo	Positivo	Negativo
Beta	Positivo	Positivo	Positivo	Positivo
Normal	Positivo	Positivo	Positivo	Negativo
Linear_quadratic	Positivo	Positivo	Positivo	Positivo
Kl_optimal	Negativo	Misto	Positivo	Negativo

Tabella 5.5: Matrice di compatibilità e stabilità visiva tra le combinazioni di Scheduler e Sampler.

Sono stati esclusi i sampler di natura ancestrale o stocastica dato che introducono una componente di rumore casuale a ogni ciclo di denoising. Iniettano una variabilità estetica incontrollabile che stravolge l'anatomia del soggetto anche a parità di seed, violando il vincolo di riproducibilità scientifica richiesto dal progetto archeologico.

5.6.2 Interpretazione Architettuale del Comportamento dei Tensori

I modelli della famiglia Flux non poggiano sulle classiche traiettorie di diffusione dei vecchi modelli U-Net, ma adottano il paradigma del Flow Matching. Questa tecnologia mappa la trasformazione dal rumore all'immagine seguendo una traiettoria geometrica rigorosamente lineare ed uniforme nello spazio latente.

Quando un sampler deterministico viene accoppiato a uno scheduler lineare, il gradiente calcola la rimozione del rumore passo dopo passo lungo una retta prefissata. Il flusso dei tensori intermedi rimane stabile e controllato, consentendo modifiche chirurgiche del prompt senza alterare l'equilibrio strutturale circostante.

Al contrario, gli scheduler complessi come Karras o Exponential sono stati matematicamente progettati per introdurre uno scaling non lineare, accelerando o decelerando la rimozione del rumore nelle fasi iniziali e finali del campionamento. Applicata al Rectified Flow di Flux.2, questa sfasatura matematica introduce una forte distorsione nel calcolo delle derivate parziali dei layer del Transformer. I valori dei tensori intermedi subiscono un'esplosione numerica che spinge il modello fuori scala, producendo immagini affette da saturazione cromatica, artefatti geometrici distruttivi o il completo stravolgimento del soggetto.

A fronte delle evidenze empiriche e analitiche raccolte, la combinazione composta dal sampler Euler associato allo scheduler Simple è stata consolidata come lo standard algoritmico ottimale dell'intero progetto. Questa accoppiata, rappresentando lo standard nativo e filologicamente più pulito per la suite Flux, garantisce il perfetto bilanciamento tra

la conservazione strutturale dei pesi LoRA addestrati su Leonardo e la plasticità di risposta alle variazioni sintattiche del prompt.



Figura 5.22: Generazione con Euler-Simple

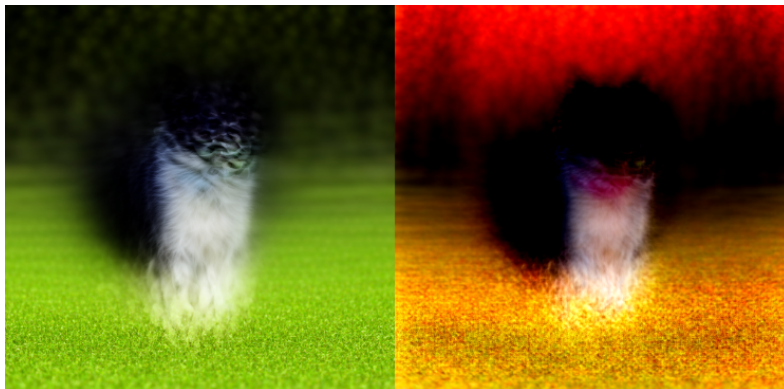


Figura 5.23: Esempio di test negativo: Euler-Karras

Capitolo 6

Analisi e utilizzo di modelli per generazione video

Se il capitolo precedente ha esplorato la strutturazione dell'identità visiva e la coerenza spaziale dei personaggi attraverso modelli di generazione di immagini e tecniche di condizionamento avanzate, questo capitolo mostra la transizione verso la dimensione temporale. L'obiettivo si sposta dalla configurazione del singolo frame statico alla gestione della fluidità del movimento, del montaggio cinematografico e della sintesi vocale adattiva, esplorando modelli di generazione Text-to-Video e Video-to-Video.

Vengono analizzate le fasi di testing locale, volte a comprendere le logiche strutturali e i limiti computazionali delle architetture di diffusione temporale, evidenziando le criticità legate alla gestione della VRAM e ai fenomeni di quantizzazione. Successivamente, si esamina la migrazione dei flussi di lavoro su architetture ad alte prestazioni grazie all'utilizzo di Leonardo, utilizzando appositi workflow all'interno di ComfyUI per i modelli LTX 2 e Wan 2.2. Infine, vengono mostrate le tecniche di rifinitura applicate, come la gestione dell'ultimo frame, la manipolazione delle espressioni facciali via LivePortrait e la sincronizzazione fonetica e acustica tramite algoritmi di sintesi vocale.

Al fine di orientare visivamente il lettore nell'articolazione della ricerca tra i diversi ambienti di calcolo e le relative pipeline software e audio

impiegate, la Figura 6.1 riassume l'architettura logica complessiva del capitolo.

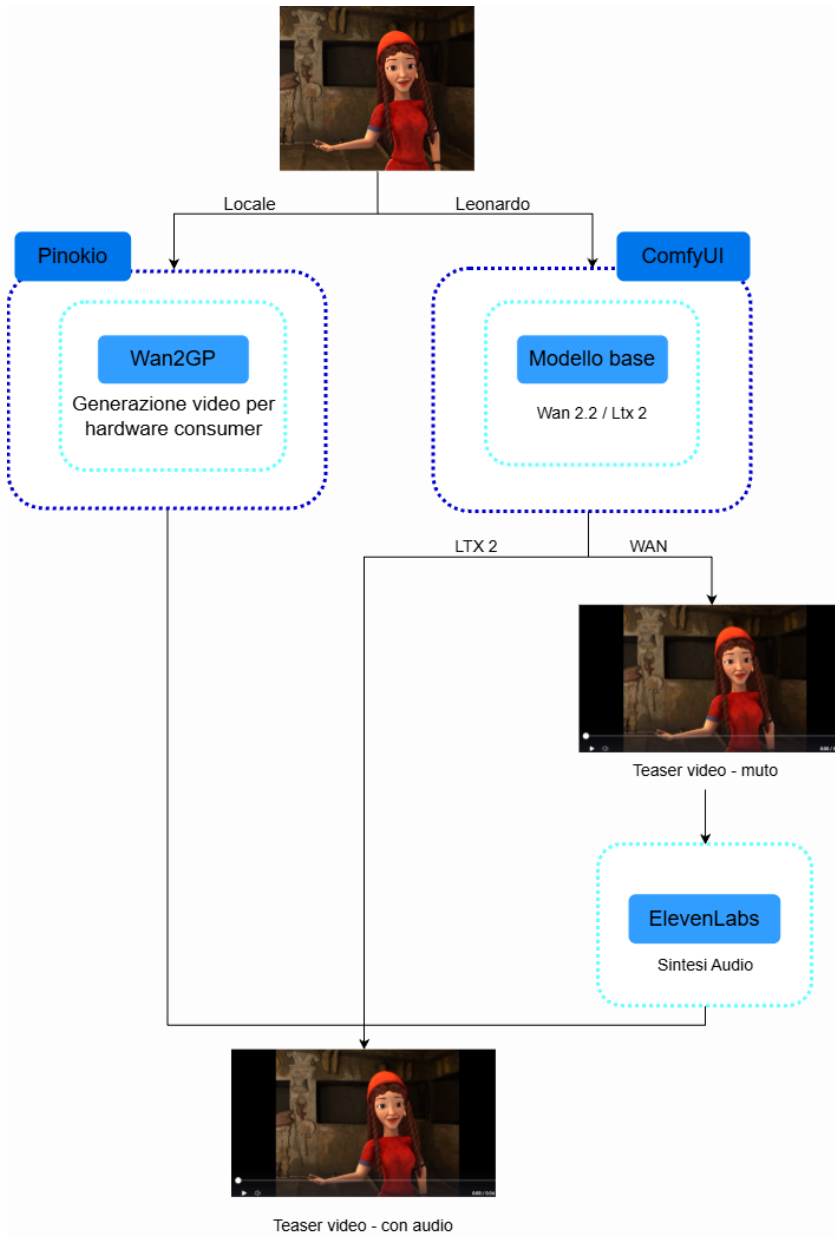


Figura 6.1: Architettura complessiva della pipeline di generazione video, ramificazione strategica tra l'ambiente di testing locale (Pinokio) e la produzione su infrastruttura HPC (ComfyUI su Leonardo) con distinzione dei flussi audio.

Come evidenziato dal workflow, l'immagine chiave viene processata seguendo due binari paralleli: la fase locale viene gestita tramite l'ecosistema Pinokio per lo screening preliminare dell'hardware, mentre la produzione massiva ad alta fedeltà su Leonardo all'interno dell'interfaccia a nodi ComfyUI. Nella parte terminale, lo schema anticipa la gestione della componente sonora, infatti, il modello multimodale LTX 2 permette una sintesi nativa del video completo di traccia, mentre l'architettura WAN genera un output muto che richiede l'integrazione esogena del modello di Text-to-Speech ElevenLabs.

6.1 Configurazione dell'ambiente locale: Pinokio

La complessità intrinseca dei modelli di generazione video basati su architetture Diffusion Transformer richiede un'infrastruttura software estremamente solida per l'orchestrazione delle dipendenze di sistema. Prima di scalare la produzione su infrastrutture HPC, si è esplorata la generazione di contenuti multimediali con hardware consumer, utilizzando Wan2GP. A causa della natura frammentata delle librerie open source e della necessità di gestione di molte dipendenze, per la gestione dell'ambiente locale si è scelto di adottare l'ecosistema Pinokio [33].

Pinokio è un ecosistema software open-source definito come un browser programmabile open-source. È progettato per operare come un computer virtuale o un micro-sistema operativo specializzato nell'orchestrazione di backend IA. La sua funzione principale consiste nell'automatizzare l'installazione, la configurazione e l'esecuzione di qualsiasi aplicação o modello di intelligenza artificiale direttamente sulla macchina locale, eliminando la necessità di intervenire manualmente sul terminale di sistema per configurare variabili d'ambiente, compilatori C++ o runtime specifici.

6.1.1 Architettura e caratteristiche

L'architettura di Pinokio si basa su un paradigma di isolamento ed esecuzione dichiarativa che offre vantaggi cruciali per la stabilità della pipeline sperimentale:

- **Scripting dichiarativo tramite JSON:** ogni applicazione all'interno dell'ecosistema viene installata e gestita attraverso script di configurazione scritti in formato JSON. Questi script descrivono l'intero ciclo di vita dell'applicazione, eseguono automaticamente il clonaggio dei repository Git, creano gli ambienti virtuali dedicati, gestiscono le istruzioni condizionali basate sul sistema operativo ospite e invocano i gestori di pacchetti senza alcuna contaminazione globale.
- **Isolamento degli ambienti condivisibili:** a differenza delle normali installazioni Python che si appoggiano a interpreti globali o a istanze Conda centralizzate, Pinokio confina ogni applicazione in una sandbox indipendente. All'interno della directory radice dell'applicazione vengono allocati file binari dedicati per Conda, Pip, Node.js e Git. Ciò garantisce che l'installazione dei requisiti di un modello video non vada a sovrascrivere o corrompere le dipendenze richieste da altre interfacce locali.
- **Interfaccia unificata e terminale Web:** Pinokio traduce i log di basso livello dei terminali di sistema in un'interfaccia grafica centralizzata. Questo permette di monitorare in tempo reale lo scaricamento dei pesi dei modelli, l'allocazione della memoria di swap e l'eventuale insorgenza di errori di compilazione, offrendo al contempo un pannello di controllo web-based per l'avvio e l'arresto dei server locali.
- **Portabilità e manutenibilità:** essendo l'intera struttura configurata come un'applicazione in cui tutte le configurazioni, i modelli scaricati e gli script risiedono in un'unica cartella principale, il

sistema permette una migrazione rapida o il backup dell'intero ambiente da un'unità di memoria a un'altra senza dover riconfigurare i registri di sistema o risolvere nuovamente i collegamenti simbolici delle librerie dinamiche.

Attraverso questa infrastruttura è stato possibile implementare l'applicazione Wan2GP, un'interfaccia Web specializzata nell'esecuzione ottimizzata dei modelli video in contesti hardware a risorse limitate, ponendo le basi per i primi test di generazione video analizzati nella sezione successiva.

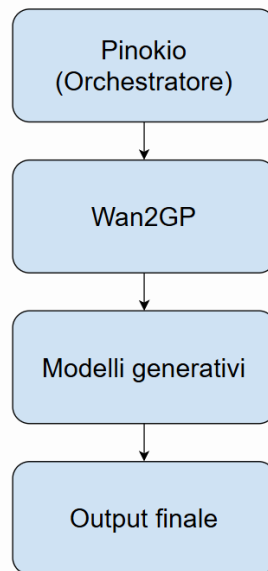


Figura 6.2: Flusso Pinokio

6.2 Analisi strutturale dei modelli generativi video open-weights

La transizione dalla sintesi d'immagine statica alla generazione di flussi video coerenti rappresenta una delle sfide più complesse nell'ambito del deep learning. Fino a tempi recenti, la generazione video si basava principalmente su architetture derivate da reti convoluzionali di tipo U-Net

che pur garantendo un’ottima fedeltà spaziale sui singoli fotogrammi, mostravano forti limiti strutturali nel mantenere la stabilità geometrica e la fluidità del movimento lungo l’asse temporale.

Per il superamento di tali barriere e per il perseguimento del rigore filologico richiesto dalla cinematizzazione del personaggio di Ati, la ricerca si è focalizzata sulle moderne architetture open-weights basate sul paradigma dei Diffusion Transformer (DiT). In questa sezione vengono analizzate nel dettaglio le caratteristiche strutturali, le dinamiche di calcolo e le divergenze architetture dei modelli selezionati per la pipeline sperimentale, ovvero Wan 2.1, Wan 2.2 e LTX 2.

6.2.1 I modelli Wan

I modelli sviluppati all’interno della famiglia Wan [48] rappresentano lo stato dell’arte nella sintesi video open-source ad alta definizione. Il cuore tecnologico di queste architetture risiede nell’integrazione dei moduli Transformer all’interno del processo di diffusione stocastica. Tuttavia, a differenza dei modelli per immagini statiche, la rete è stata interamente riprogettata e ottimizzata per scalare in modo nativo sulla dimensione temporale, operando una fusione matematica tra spazio e tempo.

L’infrastruttura di Wan si articola su tre macro-moduli interconnessi:

- **Il Text Encoder:** la comprensione linguistica viene affidata all’architettura encoder T5-XXL. Questo modulo consente di mappare istruzioni testuali stratificate, superando i colli di bottiglia dei vecchi vettori CLIP. Nel caso di Wan, l’encoder non si limita a interpretare la natura semantica del soggetto, ma decodifica accuratamente i token di movimento e le direttive legate alla cinematografia virtuale.
- **Il motore DiT spazio-temporale (Spatio-Temporal Transformer):** Wan implementa il framework del Flow Matching, nello specifico, traiettorie lineari basate su Rectified Flows), ottimizzando la convergenza del denoising. Per ancorare i token visivi alle

loro coordinate fisiche e cronologiche, la rete implementa un'estensione tridimensionale degli embedding posizionali rotanti, denominata 3D-RoPE (3D Rotary Positional Embedding), che attenua errori geometrici delle strutture durante le animazioni complesse.

- **L'Autoencoder (3D Causal VAE):** a differenza dei VAE tradizionali che comprimono i fotogrammi come matrici 2D isolate, Wan adotta un codificatore automatico causale tridimensionale. Questo modulo comprime il flusso video sia nello spazio (altezza e larghezza) sia nel tempo (durata e frame rate) applicando un fattore di sottocampionamento spaziale pari a 8 e un fattore di sottocampionamento temporale pari a 4. Il video viene così sezionato in macro-patch tridimensionali, linearizzate in una sequenza di token visivo-temporali.

I meccanismi di attenzione della rete operano simultaneamente sull'intera sequenza di token, consentendo al modello di calcolare la traiettoria di un pixel conoscendo contemporaneamente lo stato geometrico dei fotogrammi passati e futuri. Questo approccio azzerà quasi completamente il fenomeno del flickering (sfarfallio) e garantisce benefici ricercati nel lavoro di tesi:

- **Costanza anatomica di Ati:** la stabilità del volto e dei tessuti del soggetto viene preservata anche durante rotazioni della testa.
- **Coerenza ambientale:** l'illuminazione e le geometrie dello sfondo cambiano coerentemente rispettando le leggi della prospettiva e della fotometria fisica.

6.2.2 Wan 2.1 vs Wan 2.2

Le sessioni la fase di test, sono emerse differenze sostanziali tra le due versioni del modello:

- **Wan 2.1 (ottimizzazione della fedeltà spaziale):** questa versione si focalizza sulla fedeltà al prompt testuale e sulla nitidezza

del singolo fotogramma. È particolarmente indicata per scene con movimenti contenuti dove la qualità del dettaglio (texture, tessuti, volti) è prioritaria.

- **Wan 2.2 (consolidamento della coerenza temporale e volumetrica):** La versione 2.2 introduce aggiornamenti architetturali critici mirati alla risoluzione delle deformazioni morfologiche. Durante l'esecuzione di movimenti complessi della telecamera virtuale o del personaggio, i modelli tradizionali tendono a subire allucinazioni visive, modificando la volumetria corporea o duplicando elementi simmetrici. Wan 2.2 mitiga drasticamente queste irregolarità attraverso un affinamento dei blocchi di attenzione temporale, garantendo che Ati mantenga una struttura geometrica stabile.

Un limite strutturale invalicabile comune a entrambe le versioni della famiglia Wan è l'assenza nativa di sintesi sonora. L'architettura calcola e restituisce esclusivamente la matrice latente video, generando un file MP4 privo di traccia audio. Ciò impone l'adozione di software o modelli esogeni per la finalizzazione del comparto acustico, come analizzato nei successivi paragrafi dedicati alla post-produzione.

6.2.3 LTX 2

LTX 2 (noto anche come LTX-Video nella sua estensione di ultima generazione) [24] introduce nel panorama dei modelli open-weights un cambio di paradigma radicale fondato sulla multimodalità nativa. A differenza della famiglia Wan, concepita per operare in un dominio puramente visivo, l'architettura di LTX 2 unifica la sintesi cinetica e la generazione acustica all'interno dello stesso spazio latente e durante il medesimo ciclo di inferenza. Il modello si attesta su una scala computazionale più agile (circa 2.5 miliardi di parametri), ottimizzata per ridurre i tempi di calcolo massivi.

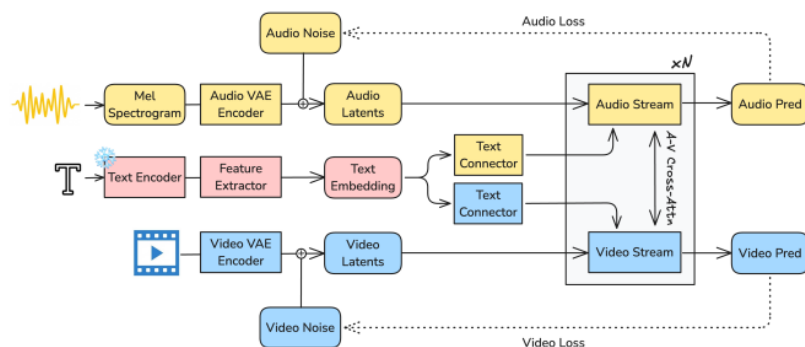


Figura 6.3: Architettura LTX 2

L'infrastruttura introduce innovazioni matematiche come:

- **L'Autoencoder Multimodale:** il modulo VAE tridimensionale di LTX 2 è progettato per comprimere simultaneamente i vettori geometrici dei fotogrammi e gli spettrogrammi della traccia audio all'interno di un unico tensore latente unificato. Spazio, tempo e suono vengono tokenizzati insieme.
- **Il flusso di inferenza parallelo:** l'architettura linguistica di LTX 2 implementa un parsing sintattico avanzato. Il modello isola le stringhe di testo inserite dall'utente all'interno di virgolette, interpretandole non come descrizioni visive dell'ambiente, ma come vettori fonetici da sintetizzare. Mentre i blocchi del Transformer calcolano l'evoluzione dei fotogrammi, strati dedicati di Cross-Attention Multi-Modale sintetizzano parallelamente la traccia vocale associata, modellando il timbro e l'intonazione in base alle direttive del prompt.

Grazie a questa unificazione matematica dei domini, LTX 2 applica in modo deterministico la funzionalità di Lip-Sync. La cinematica dei muscoli facciali, il movimento delle labbra e l'apertura della bocca di Ati vengono calcolati in diretta corrispondenza dei fonemi emessi dalla traccia audio interna. Questo approccio elimina la necessità di ricorrere a soluzioni esterne per la parte audio, sebbene la resa fotorealistica

complessiva delle texture ambientali rimanga qualitativamente inferiore rispetto ai livelli geometrici raggiunti da Wan 2.2.

6.3 Wan2GP

Per l'interfacciamento e la validazione preliminare dei modelli in ambiente locale è stata utilizzata l'applicazione Wan2GP [17]. Si tratta di un'infrastruttura software basata sul framework Gradio, esplicitamente ingegnerizzata per aggregare molteplici pipeline di generazione video avanzate all'interno di un'unica dashboard unificata.

Lo scopo della fase preliminare in locale è di quantificare la sostenibilità computazionale dei modelli di generazione video su hardware consumer ed effettuare uno screening qualitativo iniziale delle risposte dei modelli alle immagini sorgente del personaggio di Ati, prima del definitivo deployment sui nodi HPC.

6.3.1 Caratteristiche e funzionalità

A differenza dell'esecuzione standard di script monolitici da riga di comando, Wan2GP opera come un orchestratore a più funzionalità. L'applicazione supporta nativamente la generazione Text-to-Video, Image-to-Video e Video-to-Video, offrendo varie opzioni di configurazione per ciascuna modalità di utilizzo in base allo scopo desiderato.

L'esecuzione di modelli a piena precisione su schede grafiche consumer incontra un ostacolo fisico insormontabile legato alla saturazione della memoria video. Per stabilizzare il sistema e prevenire errori di allocazione (Out-of-Memory), in Wan2GP sono state implementate tre strategie algoritmiche di ottimizzazione della memoria:

- **CPU offloading sequenziale:** questa tecnica seziona l'architettura del Transformer in blocchi di layer discreti. Durante la fase di inferenza (forward pass), solo i blocchi e i tensori direttamente coinvolti nel calcolo matematico corrente vengono mantenuti in

VRAM. I layer non attivi o adiacenti vengono temporaneamente migrati (effettuando un offloading) sulla RAM di sistema della macchina host, per poi essere richiamati tramite bus PCIe solo al momento dell'esecuzione. Questo meccanismo riduce drasticamente il picco di occupazione della memoria grafica, pur introducendo colli di bottiglia nei tempi di trasferimento dei dati.

- **Tiled VAE (Variational Autoencoder) Processing:** la decodifica delle matrici latenti tridimensionali nel dominio dei pixel reali rappresenta una delle fasi a più alto assorbimento di memoria dell'intero ciclo generativo. Il modulo Tiled VAE aggira il problema frammentando l'intero blocco latente (spazio-temporale) in piccoli tasselli tridimensionali bidimensionali (tiles). La decompressione e la ricostruzione del video avvengono calcolando un tassello alla volta e assemblandoli successivamente tramite algoritmi di fusione dei bordi, impedendo la saturazione della memoria anche a risoluzioni elevate.
- **Precisioni ridotte e quantizzazione avanzata:** l'applicazione consente l'esecuzione dei modelli convertendo i pesi tensoriali originali (FP16) in formati quantizzati a precisione ridotta, quali l'8 bit (FP8 Scaled) o varianti distillate a 4 e 3 bit. La compressione numerica riduce le dimensioni fisiche del modello sul disco e in memoria, abbassando la barriera hardware d'ingresso.

6.3.2 Test operativi e limiti qualitativi

La validazione empirica è stata avviata sfruttando la sezione Guides integrata nell'interfaccia, una matrice di configurazione e pre-impostazione che mappa i parametri ottimali associandoli direttamente alle schede di esecuzione pre-configurate. Durante i test eseguiti tentando di forzare l'inferenza del modello Wan 2.1 a piena precisione matematica, l'hardware locale ha riscontrato un sistematico fallimento hardware dovuto al superamento dei limiti fisici della memoria grafica. Per rendere possi-

bile l'inferenza si è reso obbligatorio il ricorso alle versioni distillate e quantizzate Q3 e Q4.

I video generati in modalità Image-to-Video a partire dai ritratti di Ati hanno confermato la fattibilità tecnica dell'operazione, ma hanno evidenziato un severo compromesso qualitativo. La forte quantizzazione numerica applicata ai pesi introduce inevitabilmente un rumore di fondo che si traduce in artefatti visivi evidenti, come repentine fluttuazioni della luminosità della scena (flickering fotometrico), micro-deformazioni instabili dello sfondo e una progressiva perdita di fedeltà stilistica rispetto all'immagine iniziale.

Nel caso di LTX 2 quantizzato, si è riscontrata l'effettiva attivazione del parlato nativo e del Lip Sync, ma l'eccessiva compressione dei pesi ha generato un output video visibilmente sgranato e privo di quella nitidezza e dettaglio del tessuto necessarie per il rigore filologico del progetto.

In seconda istanza, sono stati esplorati workflow più complessi in modalità Video-to-Video (V2V), integrando moduli di Control Video. Questa architettura teoricamente consente di guidare lo sviluppo temporale iniettando maschere di controllo strutturali estratte da clip video reali utilizzate come traccia guida. Nei test locali è stato implementato il condizionamento a doppio frame limite (Keyframing iniziale e finale), inserendo sia il primo fotogramma che l'ultimo della sequenza, facendo generare al modello l'interpolazione geometrica della porzione centrale (Inbetweening).

Il video sintetizzato in questo modo ha mostrato un rapido collasso della coerenza e dei dettagli nella porzione centrale della transizione, manifestando gravi imprecisioni, in particolare nella gestione dell'illuminazione ambientale e della costanza dei tratti somatici di Ati.

Sotto il profilo puramente computazionale, la pipeline locale ha evidenziato grossi colli di bottiglia. Una singola generazione ottimizzata a bassa risoluzione e di breve durata ha richiesto tempi di elaborazione oscillanti tra i 30 e i 60 minuti, superando i 90 minuti per risoluzioni più elevate o per l'attivazione di configurazioni multi-layer. Questo scenario ha confermato l'impossibilità di validare e raffinare i prompt in

tempi agili in ambiente consumer. Tali risultanze sperimentali hanno reso indispensabile la migrazione dell'intera pipeline sull'infrastruttura di calcolo ad alte prestazioni HPC Leonardo, implementando i workflow basati su modelli non distillati ed esecuzione a piena precisione all'interno di ComfyUI.

6.4 Generazione su Leonardo: LTX 2

Le criticità riscontrate nei test locali hanno confermato la necessità di spostare la produzione sull'infrastruttura HPC Leonardo. Sono stati utilizzati workflow personalizzati su ComfyUI per sfruttare la potenza computazionale delle GPU del cluster, permettendo l'esecuzione di modelli non distillati a piena precisione e garantendo così la qualità necessaria agli standard cinematografici.

Si è iniziato da una prima fase di sperimentazione per capire quale modello portasse ai risultati migliori, in modo da utilizzare quello definitivo per la produzione di vari video connessi tra loro allo scopo di realizzare un filmato finale di buona durata. Lavorare su Leonardo ha permesso di utilizzare il modello LTX 2 alla sua massima potenza, con i relativi text encoders e VAE compatibili. Il workflow Image to Video (I2V) implementato è composto da nodi differenti, raggruppati in base alle funzionalità che gestiscono.

Un esempio semplice di prompt utilizzato in questo workflow è:

“The woman in a red dress presents herself saying "Ciao mi chiamo Ati e sarò una delle guide in questa mostra" while looking to the camera”

6.4.1 Impostazioni Video

Una volta caricati i modelli e l'immagine di partenza, si definiscono le impostazioni geometriche e temporali del video che si vuole generare. Il nodo Get Image legge i metadati dell'immagine di input e passa i valori

di `width` (larghezza) e `height` (altezza) ai nodi successivi, in modo da garantire che il video finale abbia proporzioni perfettamente coerenti con l'immagine di partenza, evitando distorsioni del personaggio di Ati.

Il nodo `EmptyImage` crea un'immagine vuota che serve unicamente a definire l'aspect ratio e la risoluzione finale su un'immagine estremamente leggera dal punto di vista computazionale. Il nodo `Upscale Image By` sfrutta l'output del nodo `EmptyImage` e utilizza l'algoritmo Lanczos, noto per la sua alta qualità nel ridimensionamento e nella conservazione dei dettagli dei bordi. Impostando il suo parametro `scale_by` a 0.5 si va a ridurre la risoluzione del video del 50%. Questo è utile per alleggerire il carico computazionale iniziale e velocizzare la generazione. Successivamente, il nodo `Get Image Size` calcola le nuove dimensioni dimezzate dal passaggio precedente e invia queste misure finali al `Sampler`. Senza questo nodo, il modello proverebbe a generare il video alle dimensioni originali del file di input, ignorando lo scaling e causando un fallimento per disallineamento della matrice. Infine, il nodo `Length` permette di definire la durata del video impostando il numero di FPS (Frame Per Second) da cui è composto.

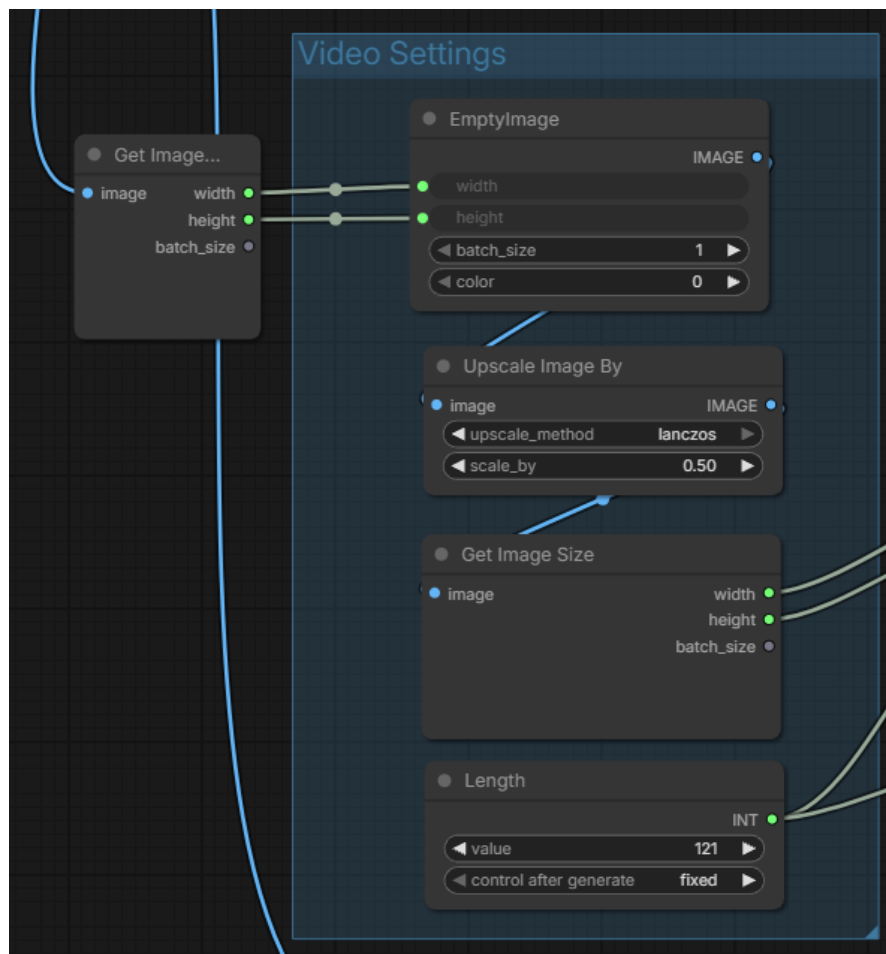


Figura 6.4: workflow LTX2 - Video Settings

6.4.2 Gestione dello spazio latente

In questa fase vengono presi in input i valori delle impostazioni video per proiettarli nel dominio della diffusione. Il nodo `LTXVPreprocess` converte i pixel dell'immagine in un formato che il modello video può sovrapporre correttamente al rumore latente. Si tratta di un parametro tecnico che definisce quanto spazio l'immagine debba occupare una volta trasformata in token. Un valore di 33 si è dimostrato bilanciato per mantenere la fedeltà filologica del personaggio senza sovraccaricare il campionatore.

A differenza del nodo `EmptyImage` visto in precedenza che lavorava nel-

lo spazio dei pixel visibili, il nodo `EmptyLTXVLatentVideo` crea lo spazio latente vero e proprio, generando un blocco di rumore tridimensionale (larghezza x altezza x tempo).

Il nodo `LTXV Empty Latent Audio` implementa la multimodalità nativa dell'architettura. Questo modulo crea infatti una traccia audio vuota della stessa durata del video che serve come binario digitale su cui il modello innesterà la voce o i suoni ambientali durante l'inferenza. Il parametro `frame_rate` definisce la velocità di sincronizzazione tra asse visivo e acustico.

Il nodo `LTXVImgToVideoInplace` inserisce l'immagine preprocessata dentro il blocco di rumore latente, dando una direzione precisa alla generazione. Impostare il parametro `strength` a 1.0 è fondamentale, in quanto si indica al modello che il primo frame del video deve essere identico all'immagine iniziale. Abbassando il valore, l'IA si prenderebbe più libertà creativa, alterando il soggetto.

Infine, il nodo `LTXVConcatAVLatents` accorpa il latente del video e quello dell'audio in un unico pacchetto multimodale, permettendo al Sampler di elaborare video e audio contemporaneamente. È grazie a questo nodo che LTX 2 riesce a sincronizzare il movimento delle labbra di Ati con il suono emesso, poiché vede entrambi i dati come un unico flusso informativo coerente.

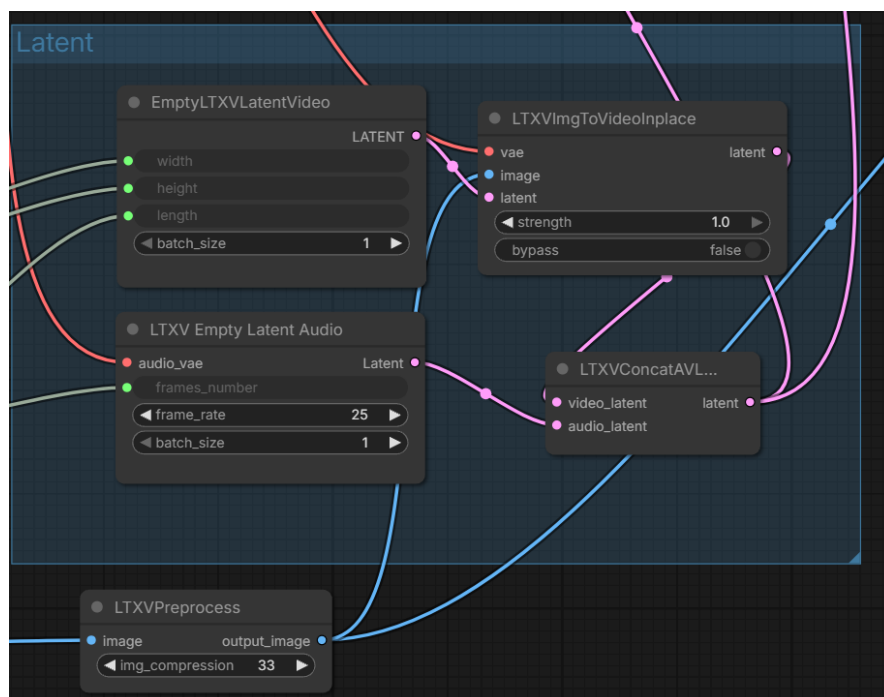


Figura 6.5: workflow LTX2 - Gestione dello spazio latente

6.4.3 Sampling

In questa fase, il rumore matematico e le istruzioni semantiche del prompt si fondono per creare i pixel del video e l'audio del personaggio. Il `SamplerCustomAdvanced` esegue materialmente il calcolo di denoising, trasformando il blocco di rumore unificato (audio + video) in un contenuto strutturato. Il nodo prende in input i parametri fondamentali (rumore, guider, sampler, scheduler, sigmas) che in questo caso vengono gestiti in maniera separata con dei nodi appositi, in modo da offrire un controllo superiore su ciascuna componente matematica del processo.

Successivamente, il nodo `LTXVSeparateAVLatent` separa nuovamente il flusso latente campionato in due binari distinti, ovvero `video_latent` e `audio_latent`.

Infine, il nodo `LTXVCropGuides` assicura che i condizionamenti del prompt positivi e negativi siano applicati correttamente solo all'area vi-

sibile del video, evitando che l'IA generi dettagli incoerenti o artefatti ai bordi dell'inquadratura di Ati.

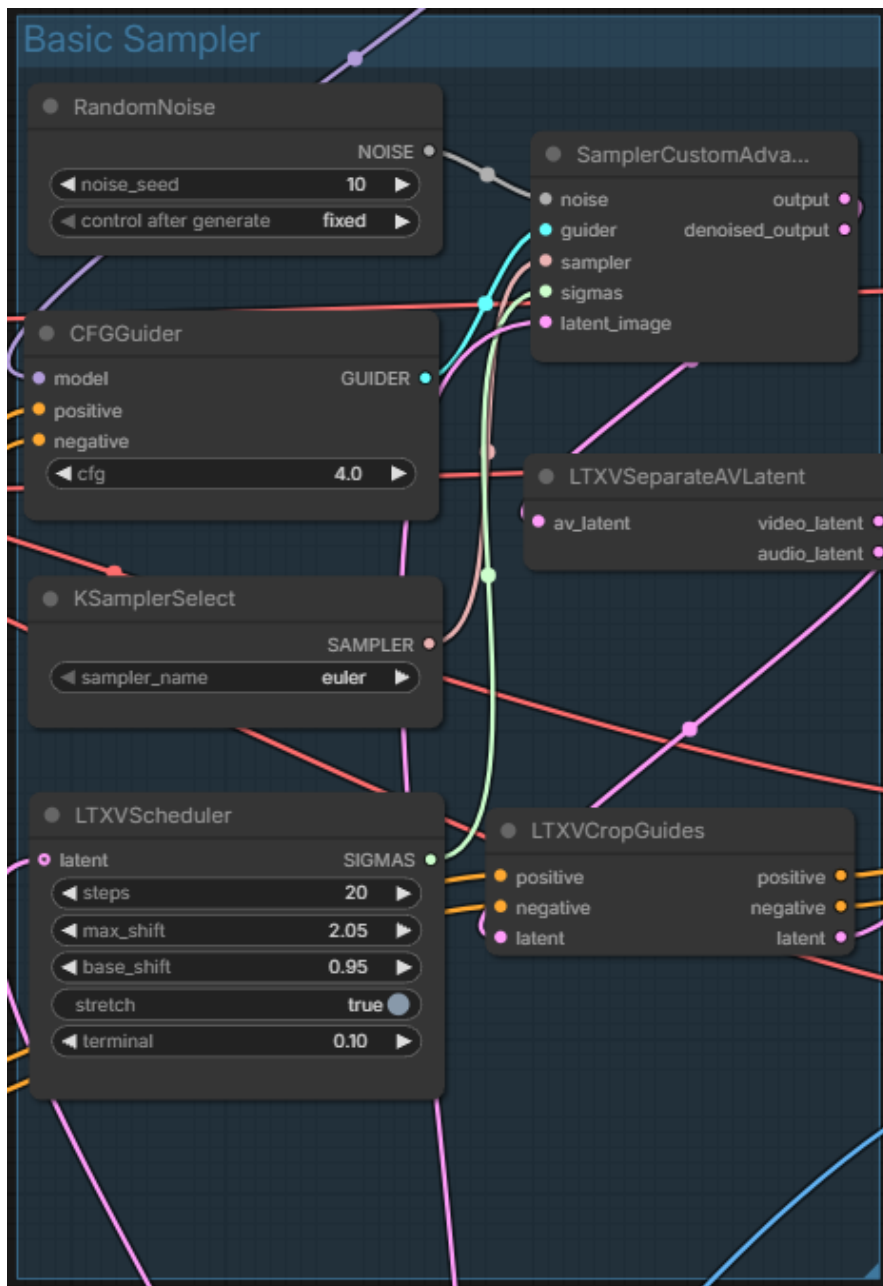


Figura 6.6: workflow LTX2 - Gestione dello spazio latente

6.4.4 Fase di Upscaling

In questo blocco il modello non deve inventare una nuova scena, ma deve ripulire e arricchire quella esistente ripristinando la risoluzione originale. Il modulo LoRA ltx-2-19b-distilled-lora-384 aiuta il modello principale a focalizzarsi sulla ricostruzione dei micro-dettagli e della texture superficiale dell'inquadratura.

Il nodo Load Latent Upscale Model carica il modello spatial upscaler-x2 il quale, a differenza di un simple ingrandimento dei pixel, lavora nello spazio latente per calcolare probabilisticamente i dettagli mancanti durante il raddoppio della risoluzione.

Il nodo Spatial rappresenta il cuore geometrico dell'upsampling. Prendendo in input i campioni latenti (samples) generati nella fase precedente per applicare l'algoritmo di upscale spaziale. Questo processo è governato dal blocco Upscale Sampling (2x), che esegue un secondo passaggio mirato di denoising ed è composto da quattro nodi specifici:

- **RandomNoise:** genera un nuovo strato di rumore sottile ad alta frequenza che serve al modello per aggiungere micro-dettagli realistici.
- **CFGGuider:** in questa fase di upscale il valore del CFG è impostato al minimo (1.0). Questo impedisce all'IA di modificare il contenuto iconografico del video, forzandola a migliorare solo la nitidezza di ciò che è già stato generato.
- **ManualSigmas:** definisce una curva di denoising molto specifica, istruendo il modello a intervenire solo sui dettagli fini senza stravolgere le forme geometriche principali.
- **KSamplerSelect (gradient_estimation):** utilizza un metodo di campionamento specializzato per l'upscale, ottimizzato per mantenere la coerenza dei bordi e delle sfumature.

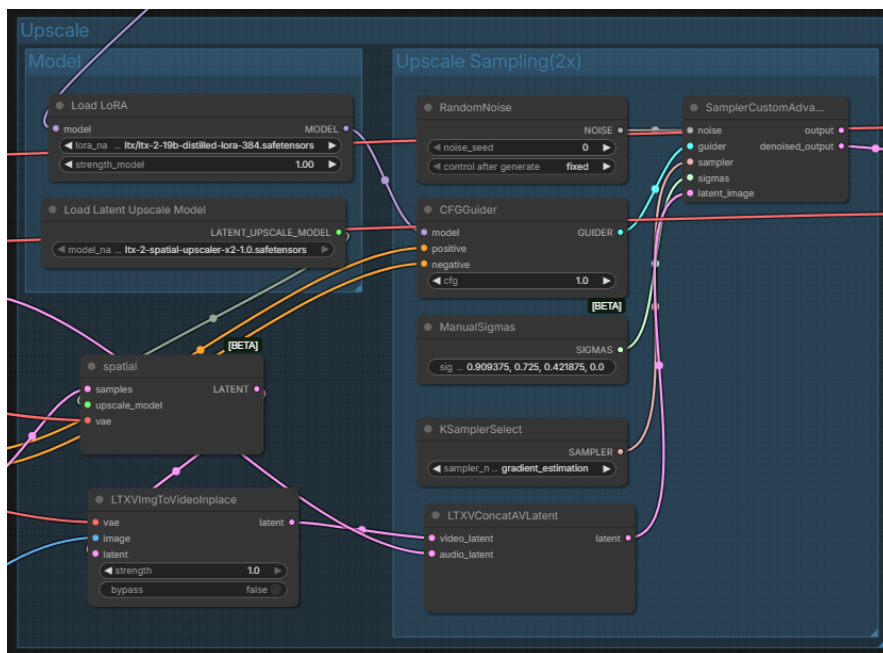


Figura 6.7: LTX 2 workflow - Upscaling

6.4.5 Movimento di Camera

Una particolarità dei workflow utilizzati per la filiera video è la presenza di modelli LoRA esplicitamente sviluppati per gestire il movimento della telecamera virtuale secondo precise traiettorie registrate. Nel caso di LTX 2 sono stati implementati due nodi Load LoRA che caricano il medesimo modello influenzando la generazione in parallelo, uno nella fase di sampling e l'altro in quella di upscaling. In base al file selezionato, il movimento viene triggerato dall'inserimento nel prompt di una specifica parola chiave. La libreria di controllo comprende diverse opzioni:

- **LTX-2-19b-LoRA-Camera-Control-Dolly-In:** zoom/carrellata in avanti.
- **LTX-2-19b-LoRA-Camera-Control-Dolly-Out:** zoom/carrellata all'indietro.
- **LTX-2-19b-LoRA-Camera-Control-Dolly-Left / Right:** movimento laterale verso sinistra o destra.

- **LTX-2-19b-LoRA-Camera-Control-Jib-Down / Up:** movimento verticale dall'alto in basso o viceversa.
- **LTX-2-19b-LoRA-Camera-Control-Static:** inquadratura fissa.

Le sessioni di test condotte attivando o escludendo tali moduli hanno dimostrato che le scene con i LoRA abilitati vengono fortemente influenzate. Il movimento di macchina si presenta fluido e privo di distorsioni prospettiche, causando al contempo un cambiamento adattivo nel comportamento di Ati. In particolare, nei movimenti Dolly Left e Right, il movimento della telecamera ha influenzato il personaggio. Infatti, Ati ha iniziato a camminare lateralmente per seguire il movimento della telecamera.

6.4.6 Generazione dell'output

La fase terminale del grafo gestisce la conversione dei dati latenti in file multimediali riproducibili. Il nodo VAE Decode prende in carico il vettore `video_latent` (la rappresentazione matematica compressa del video) e lo decompone attraverso le matrici del Variational Autoencoder, trasformando i dati latenti in una sequenza di immagini reali (pixel) che compongono il video di Ati.

Il nodo LTXV Audio VAE Decode compie la stessa operazione speculare per la parte audio.

Successivamente, il modulo Create Video riceve i fotogrammi dal VAE visivo e la traccia sonora dall'Audio VAE Decode, unendo fisicamente le immagini e il suono in un unico file video integrato, il quale viene infine serializzato dal nodo Save Video nella cartella di destinazione.

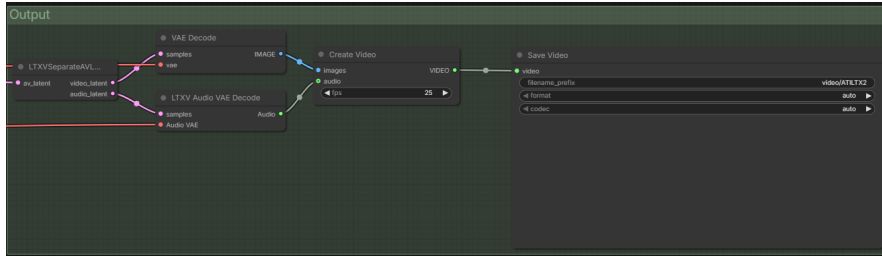


Figura 6.8: LTX 2 workflow - Output

6.4.7 Risultati

I risultati ottenuti con LTX 2 si sono rivelati estremamente validi sotto il profilo dell'aderenza al prompt, del dettaglio nei movimenti e della disposizione degli elementi archeologici in scena. Tuttavia, l'analisi degli output ha fatto emergere problemi strutturali in molte generazioni.

Al di là di una prevedibile quota di rumore in alcune transizioni molto rapide, LTX 2 ha mostrato una tendenza sistematica al fotorealismo eccessivo, alterando i tratti caratteristici del personaggio nel tentativo di rendere l'output visivo simile a una produzione cinematografica reale. L'architettura del modello, essendo stata addestrata massicciamente su video live-action ad altissima definizione, ha come obiettivo primario la verosimiglianza biologica. Questo aspetto, entrando in diretto conflitto con la natura del modello digitale di Ati, ha determinato l'incompatibilità del modello con gli scopi della tesi, portando alla scelta definitiva di adottare il modello Wan 2.2.

6.5 Generazione su Leonardo: Wan 2.2

Per l'effettiva generazione dei contenuti video della filiera produttiva si è scelto di adottare il modello Wan 2.2 sul cluster Leonardo. Questa decisione è scaturita da un bilanciamento ottimale tra tre fattori chiave quali la velocità di calcolo, l'elevata aderenza all'immagine iniziale e la qualità complessiva dell'output che ha permesso di ottenere la massima fedeltà alle caratteristiche *cartoon* del personaggio, a differenza di LTX

2. L'obiettivo metodologico risiede nella generazione di una serie di clip brevi, destinate a essere successivamente assemblate in fase di montaggio per comporre un filmato più articolato e di lunga durata. All'interno del video finale, il personaggio di Ati indica un ventaglio rituale all'interno della tomba etrusca di Cerveteri, illustrandone le caratteristiche e la storia direttamente allo spettatore.

Inizialmente sono state svolte varie prove di generazione a partire da immagini statiche precedentemente sintetizzate, le quali hanno definito una solida base metodologica per le successive generazioni video che hanno poi portato al filmato finale. Successivamente, data la presenza di video già esistenti realizzati in Blender a cui questo nuovo contenuto doveva agganciarsi (anch'essi esposti alla mostra di San Francisco), come frame iniziale è stata utilizzata un'immagine di Ati già presente in formato .png, sovrapposta a due render fatti ad hoc della Tomba Etrusca per selezionarne le esatte luci e prospettive, definendo le inquadrature in modo preciso. Con questo metodo sono state selezionate due inquadrature, una per la parte di presentazione dell'avatar digitale e una in cui Ati parla accanto a un ventaglio etrusco presente all'interno della tomba. È proprio questo ventaglio l'oggetto archeologico centrale da valorizzare, di cui l'avatar fornisce una spiegazione allo spettatore.



Figura 6.9: Frame iniziali del video



Figura 6.10: Frame intermedi del video

Un prompt utilizzato per generare una delle scene, nello specifico quella in cui Ati indica il ventaglio, è:

“The woman in the red dress starts speaking naturally, her mouth moving in sync with an unheard conversation for several seconds with a happy expression while looking at the camera and pointing at the fan on the wall behind her to the left of the image. She is constantly looking at the camera. Finally, she stops talking, keeping her mouth closed. In the end she has her lips closed and a relaxed expression.”

Dati i limiti strutturali dei modelli DiT nella gestione dell’audio, l’ingegnerizzazione del prompt ha richiesto l’inserimento di una intenzionale ridondanza semantica per esplicitare la volontà di terminare il parlato verso la fine del video. Questo vincolo testuale ha imposto al modello una transizione forzata verso uno stato di riposo espressivo negli ultimi fotogrammi della clip, facilitando poi la sovrapposizione dell’audio in fase di montaggio.

6.5.1 Workflow principale

Per l’esecuzione di Wan 2.2 nella sua versione da 14 miliardi di parametri (14B) è stata implementata la configurazione con precisione quantizzata FP8 Scaled. Questa scelta si è rivelata cruciale per ottimizzare l’allocazione delle risorse sulla GPU del cluster, garantendo una generazione

fluida ed evitando i colli di bottiglia della memoria video. I test comparativi svolti dalla community online hanno dimostrato che la compressione in FP8 non introduce una degradazione apprezzabile ad occhio nudo della qualità visiva rispetto allo standard nativo in FP16. Il formato FP8 Scaled si è dimostrato particolarmente efficace nel preservare l'integrità stilistica e la coerenza dei tratti di Ati, riducendo drasticamente i tempi di calcolo (del 30–50%) e garantendo una superiore stabilità del sistema durante le sessioni di inferenza massiva.

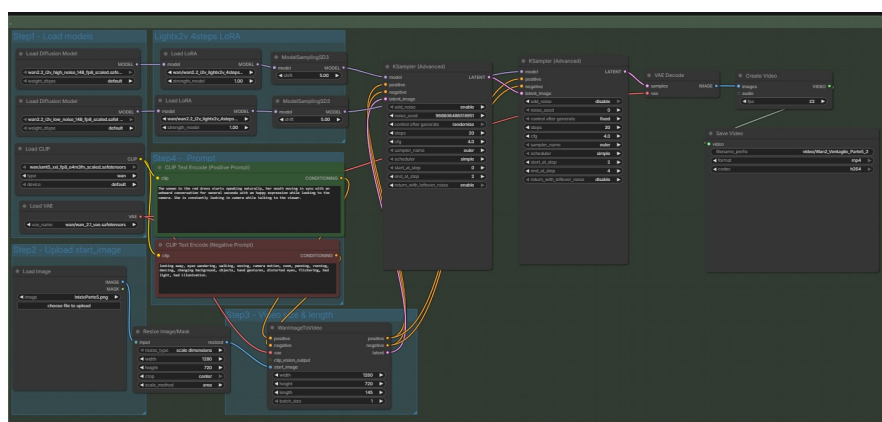


Figura 6.11: WAN 2.2 - workflow principale

Una delle particolarità più significative di questo workflow risiede nell'adozione di due versioni ottimizzate dello stesso modello, applicate a fasi differenti del processo di denoising. Questo approccio duale permette di ottenere un output sensibilmente più pulito e fedele rispetto all'utilizzo di un modello generico:

- **Modello High Noise:** è ottimizzato per intervenire nelle prime fasi del campionamento, ovvero quando il canvas latente è ancora dominato dal caos e dal rumore matematico. Il suo compito è interpretare l'immagine di input e il prompt semantico per stabilire la struttura macroscopica del video, definendo la composizione della scena e le animazioni. A livelli elevati di rumore, il modello è progettato per isolare le forme globali senza farsi distrarre dai dettagli minuti.

- **Modello Low Noise:** entra in funzione all'interno del secondo modulo KSampler (Advanced), quando la struttura geometrica del video è già consolidata ma appare ancora sgranata. Questo modello si focalizza esclusivamente sulla coerenza delle texture superficiali, sulla nitidezza degli occhi del personaggio e sulla fluidità fisica dei tessuti dei vestiti. Essendo addestrato per non alterare le forme preesistenti, il suo intervento si limita a ripulire i pixel residui e a rifinire il dettaglio fine.

La curva di rimozione del rumore viene gestita analiticamente tramite il nodo ModelSamplingSD3, che agisce sul processo attraverso il parametro di shift. Modificando questo valore, è possibile determinare se il modello debba dedicare una maggiore porzione del tempo di calcolo alla definizione della struttura globale o alla rifinitura dei dettagli geometrici. Lo shift modifica a livello matematico la percezione del tempo del modello durante il campionamento, secondo tre finestre operative:

- **Shift Basso (1.0–3.0):** il modello introduce i dettagli in modo estremamente rapido. Si tratta dell'impostazione ideale per risoluzioni ridotte, dove lo spazio di manovra latente per il calcolo è strutturalmente limitato.
- **Shift Medio (4.0–6.0):** rappresenta il punto di equilibrio ottimale tra coerenza strutturale e micro-dettaglio. Questa configurazione è stata adottata come standard per le generazioni in alta definizione, infatti, nelle generazioni svolte è stato impostato a 5.
- **Shift Alto (7.0–10.0):** Forza il modello a lavorare a lungo sulla forma generale prima di definire i dettagli fini. Viene impiegato per risoluzioni molto elevate o come contromisura per mitigare artefatti visivi, distorsioni e fenomeni di sfarfallio.

Per accelerare i tempi di produzione, nel workflow sono stati integrati due moduli LoRA specifici per le fasi di high e low noise. Normalmente, un'architettura di tipo Diffusion Transformer richiede tra i 20 e i 50 passi

per eliminare il rumore. L'attivazione di questi LoRA permette al sistema di prevedere la convergenza del risultato finale in modo molto più efficiente, consentendo di ridurre l'intera generazione a soli 4 step totali, senza compromettere la coerenza del personaggio. A differenza dei comuni metodi di accelerazione, questi moduli per Wan 2.2 sono strutturalmente trasparenti, non sovrappongono uno stile esogeno all'immagine, ma accelerano l'algoritmo nativo preservando la fedeltà filologica dell'asset di partenza.

Per la quasi totalità delle generazioni questi LoRA sono stati abilitati. Tuttavia, per alcune generazioni a risoluzioni più elevate è stato necessario rimuovere i LoRA e impostare 30 steps di denoising in modo da speculative su un risultato di qualità maggiore, soprattutto per evitare flickering di illuminazione durante le parti centrali del video.

Il caricamento e la mappatura dell'immagine avvengono tramite il nodo WanImageToVideo (corrispettivo funzionale dell'unità EmptyLTXVLatentVideo esplorata in LTX 2) che proietta i pixel nello spazio latente tridimensionale.

Il processo si articola poi sui due KSampler in cascata. All'interno del primo KSampler, un parametro molto importante in questo caso è `return_with_leftover_noise`, esplicitamente abilitato. Se questa opzione venisse disattivata, il KSampler imporrebbe il valore finale di sigma a zero, forzando la completa de-rumorizzazione della latenza di output già al primo passaggio. Questo comportamento è desiderabile solo quando il campionatore rappresenta l'ultimo anello della catena di generazione, mentre in questo caso interromperebbe il flusso di raffinamento del secondo modulo. Al termine della catena, il video viene ricostruito e salvato tramite il nodo Save Video.

6.5.2 Esplorazione dei workflow sperimentali

In fase di ricerca sono stati testati diversi workflow sperimentali dotati di funzionalità alternative, volti a semplificare specifiche fasi di lavoro e valutati in base ai rispettivi vantaggi e svantaggi computazionali.

Nella maggior parte delle architetture alternative, la variazione principale risiede nel tipo di nodo deputato all'invio dello spazio latente al campionatore.

Un primo caso studio è rappresentato dal nodo WanFirstLastFrameToVideo. Questo modulo consente di fornire in input due immagini di riferimento distinte, che vanno a vincolare rigidamente il primo e l'ultimo fotogramma della clip generata.

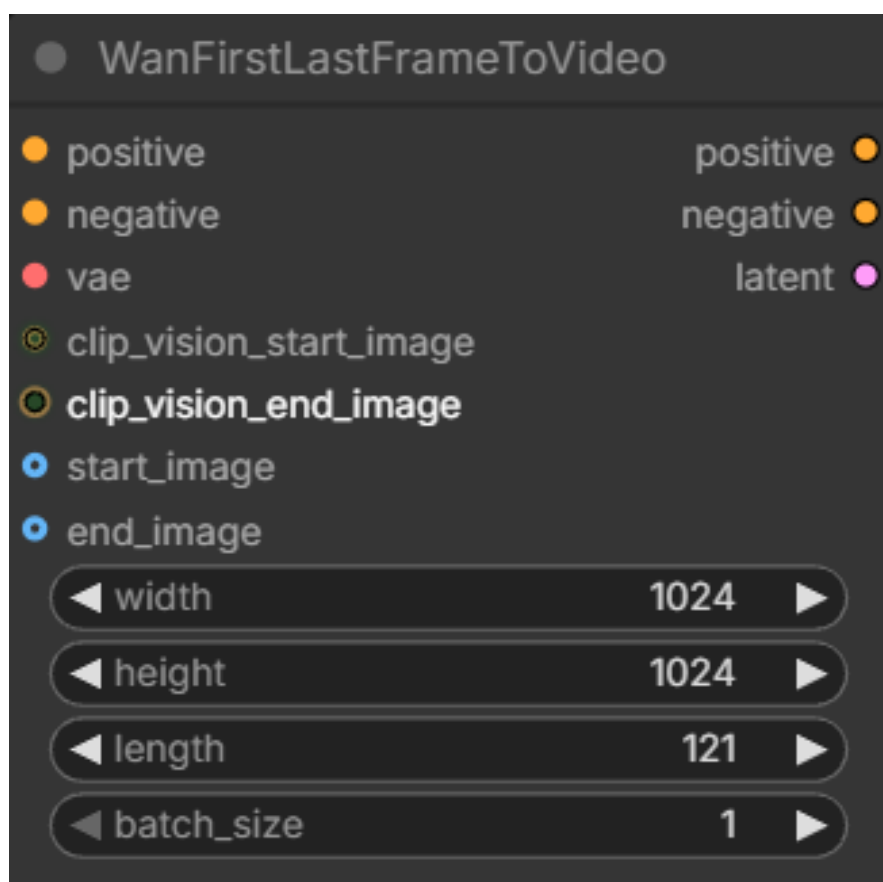


Figura 6.12: Nodo WanFirstLastFrameToVideo

Sebbene questo approccio semplifichi la gestione delle transizioni e della chiusura delle scene, i test sul campo hanno evidenziato una gestione qualitativamente inferiore della coerenza visiva e del livello di dettaglio rispetto al nodo standard WanImageToVideo, mostrando criticità

vistose soprattutto nel mantenimento della costanza dell'illuminazione ambientale.

Per quanto riguarda il controllo della telecamera virtuale, Wan 2.2 permette di governare i movimenti registrati senza ricorrere a LoRA esterni, delegando il compito al nodo nativo WanCameraEmbedding (Figura 5.9).

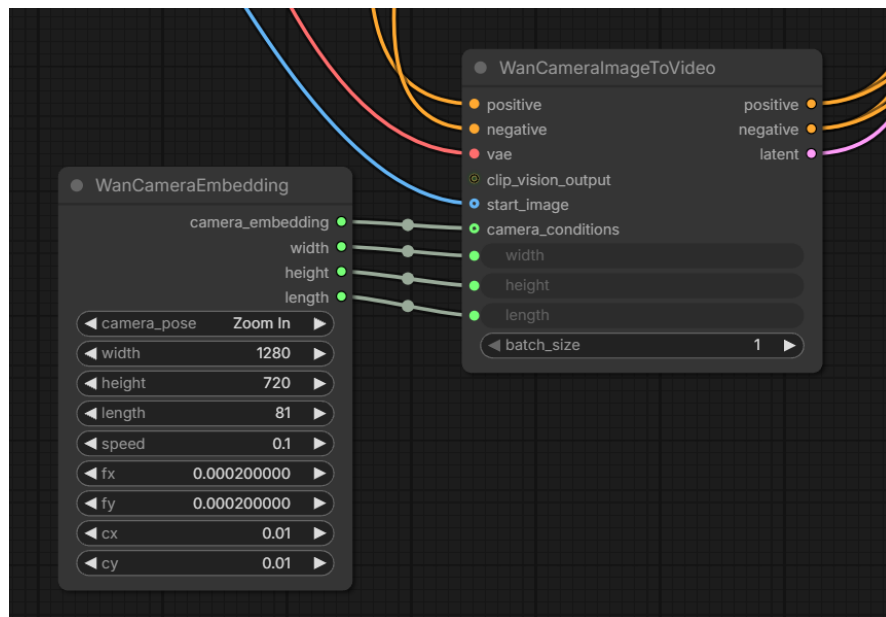


Figura 6.13: WAN 2.2 - gestione della telecamera

Attraverso i parametri interni di questo modulo è possibile configurare analiticamente la cinematica della scena impostando la traiettoria (**camera_pose**), le dimensioni dell'output, la lunghezza della traiettoria (**length**), la velocità di spostamento (**speed**), i parametri della lente focale (**fx** e **fy**) e le coordinate del punto principale dell'inquadratura (**cx** e **cy**). L'output di questo blocco viene poi inviato direttamente al nodo WanCameraImageToVideo per la generazione dello spazio latente condizionato.

Un secondo filone di sperimentazione ha riguardato l'integrazione nativa dell'audio all'interno di un unico grafo ComfyUI, utilizzando il template basato sul nodo WanSoundImageToVideo.

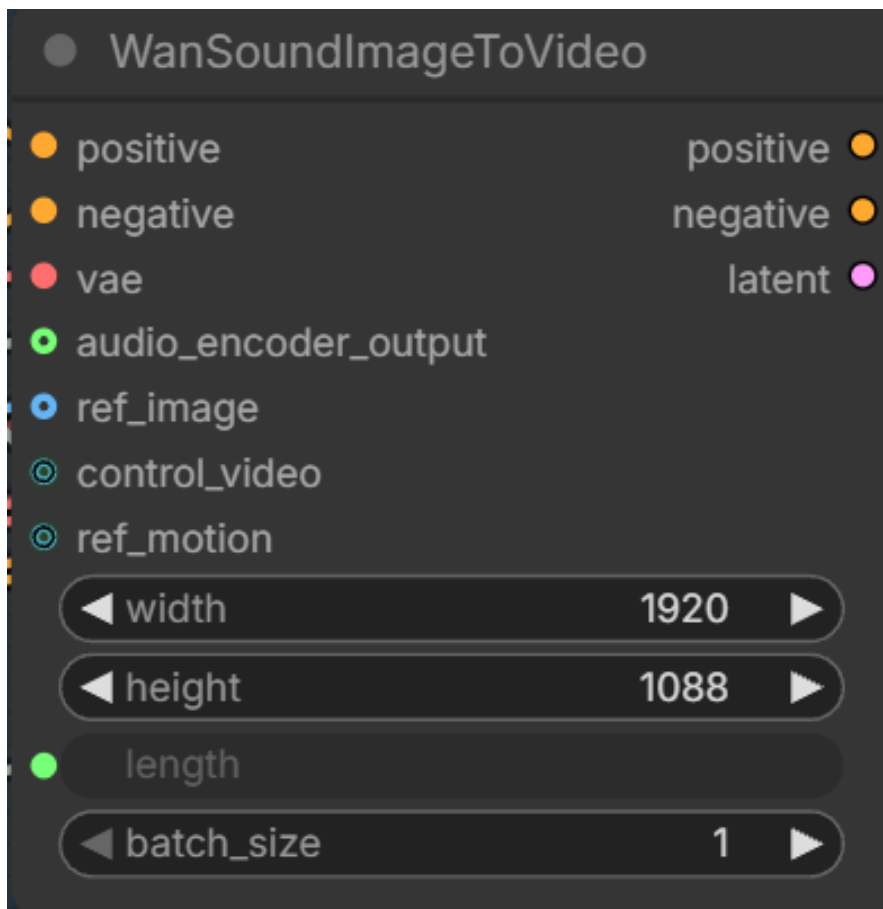


Figura 6.14: WAN 2.2 - gestione audio

Questo modulo accetta simultaneamente l'immagine di riferimento e una traccia acustica (caricata tramite il nodo LoadAudio), processando le informazioni in modo sincrono. Una particolarità di questa variante è la gestione dell'estensibilità temporale, invece di aumentare linearmente il numero di fotogrammi iniziali, il video viene esteso progressivamente attraverso nodi di espansione dedicati.

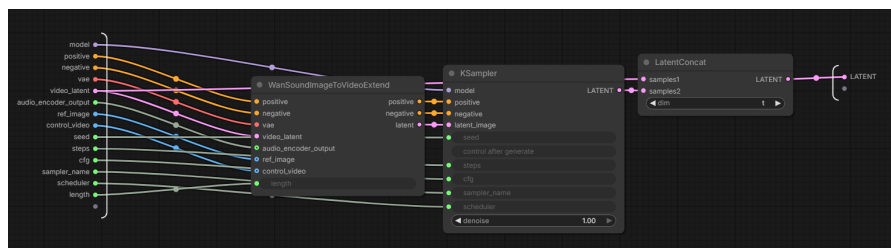


Figura 6.15: WAN 2.2 - Estensione video con audio

Il modulo WanSoundImageToVideoLatent funge da ponte verso le fasi successive, il KSampler genera i nuovi fotogrammi basandosi sulle istruzioni acustiche e il nodo LatentConcat unisce fisicamente la nuova porzione latente a quella precedente. Reiterando questa sequenza di tre elementi è possibile estendere il video alla durata desiderata. Tuttavia, i risultati sperimentali hanno evidenziato un decadimento della qualità visiva rispetto al workflow base, una scarsa precisione nella sincronizzazione labiale e un'evidente perdita di coerenza geometrica nei frame finali della catena.

Infine, sono stati condotti test con i sistemi di Control Video per guidare i movimenti del soggetto o trasferire lo stile di un video sorgente sull'immagine di Ati, sfruttando il nodo Wan22FunControlToVideo.

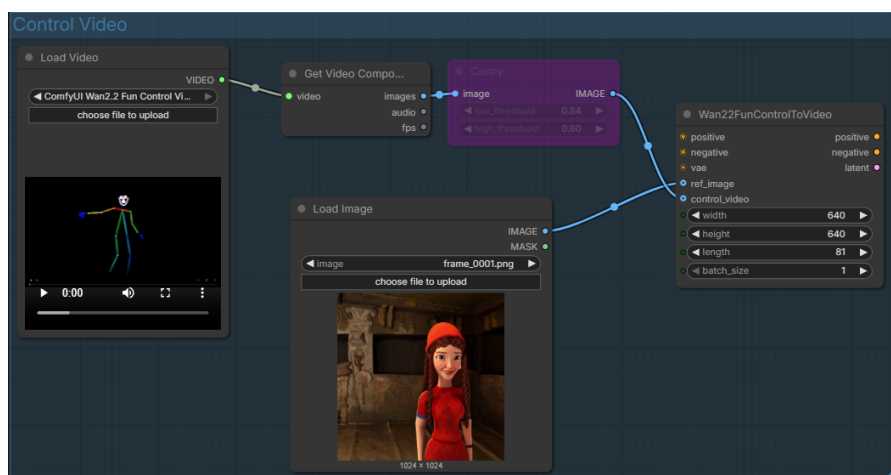


Figura 6.16: WAN 2.2 - Control Video

Le prove effettuate non hanno restituito livelli di dettaglio accettabili per gli standard richiesti dalla tesi. Il flusso prevedeva l'impiego del preprocessore grafico Canny per l'estrazione dei contorni lineari del video di guida. In alternativa, l'integrazione di un preprocessore di tipo Pose avrebbe consentito l'estrapolazione diretta dello scheletro dei movimenti. Sebbene questa tecnologia sia stata esclusa dalla produzione corrente a causa della maturità dei dettagli, essa rappresenta una solida base di ricerca per futuri sviluppi nel controllo vincolato della generazione.

6.5.3 Gestione dell'ultimo frame

Poiché l'architettura standard di Wan 2.2 non elabora l'audio in modo nativo, si sono verificate situazioni in cui clip di ottima qualità visiva terminavano in posizioni non idonee al montaggio (ad esempio con il personaggio di Ati con gli occhi semichiusi o la bocca aperta). Per risolvere tali incoerenze e prolungare le clip di un secondo è stato sviluppato un workflow di rifinitura finale basato sulla manipolazione dell'ultimo fotogramma utile della generazione precedente.

Per implementare questa soluzione sono stati installati i nodi custom `AdvancedLivePortrait` e `VideoHelperSuite`. A causa di alcune incompatibilità software riscontrate con l'ambiente di calcolo di Leonardo, questa specifica fase di post-produzione è stata interamente eseguita in locale.

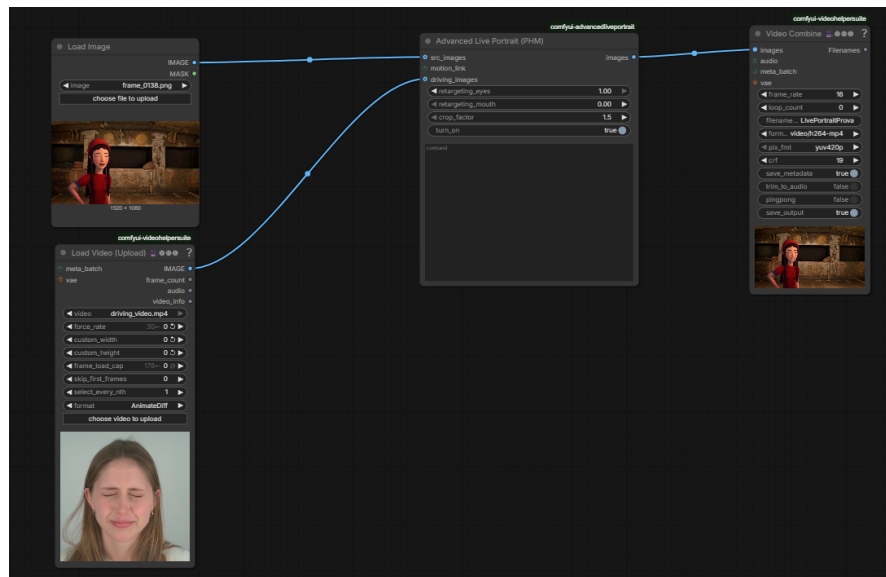


Figura 6.17: Live Portrait - modifica espressione tramite video

Un primo metodo di utilizzo di queste suite prevede l'animazione dell'immagine statica guidata dai movimenti facciali estratti da un video target di una persona reale. Sebbene l'approccio risulti intuitivo, i test hanno mostrato una parziale perdita di controllo geometrico, poiché la topologia del modello 3D di Ati non sempre si adatta con precisione millimetrica alla fisionomia umana del video guida.

Per ovviare a questo limite, si è optato per un approccio rigoroso basato sull'uso sequenziale dei nodi di editing delle espressioni (Expression Editor), che consentono la manipolazione controllata di un singolo fotogramma alla volta.

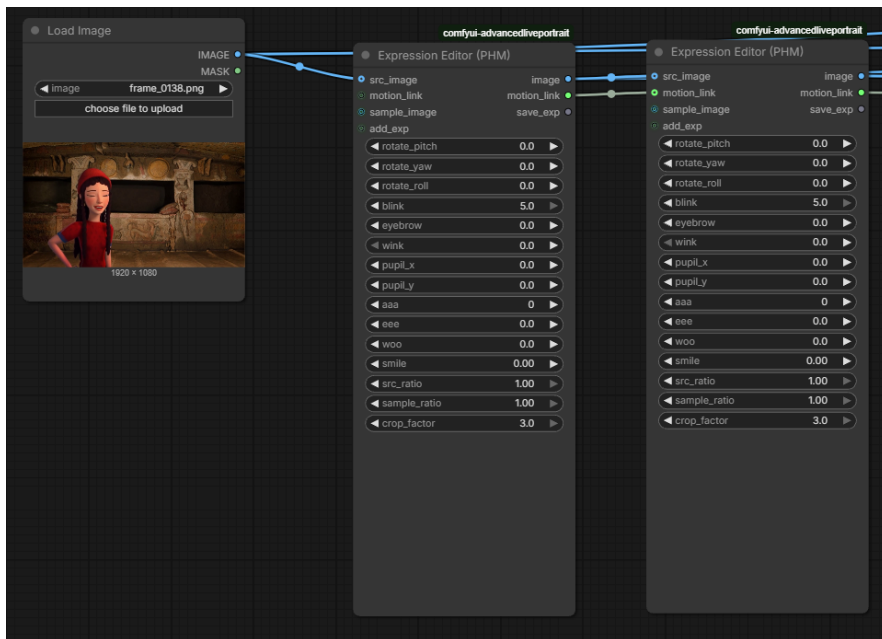


Figura 6.18: WAN 2.2 - Expression Editor

Inserendo l'immagine di partenza nel primo nodo, l'unità restituisce un fotogramma modificato in base a parametri matematici precisi, che permettono di regolare la direzione dello sguardo, l'apertura delle palpebre, la conformazione della bocca, la posizione del mento e l'area di influenza della deformazione facciale. Concatenando in serie un numero di nodi Expression Editor esattamente pari al numero di FPS necessari per coprire il secondo di estensione e inviando i fotogrammi risultanti ai nodi Create Video e Save Video, si ottiene un'animazione fluida e totalmente controllata. Questo approccio, pur richiedendo una calibrazione manuale e complessa, garantisce il controllo assoluto su ogni singolo frame di transizione.

6.5.4 Valutazione dei workflow audio in ComfyUI

Nel tentativo di centralizzare l'intera produzione all'interno dell'interfaccia ComfyUI ed evitare il ricorso a piattaforme esterne, sono stati testati

alcuni grafi di sintesi vocale nativi basati sulle librerie TTS Audio Suite e ComfyUI_Chatterbox_SRT_Voice.

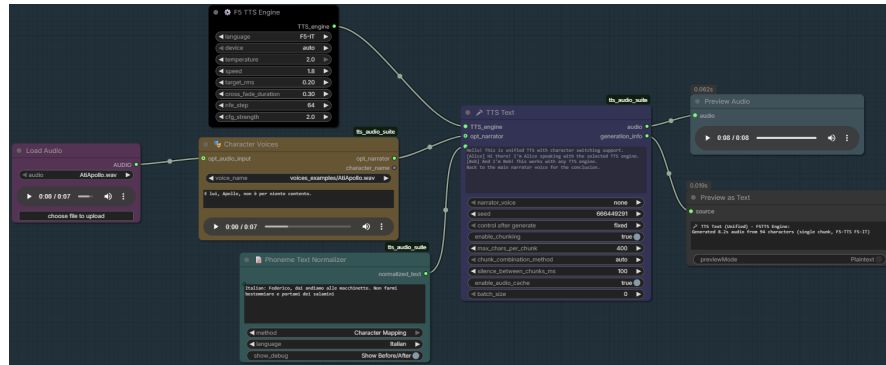


Figura 6.19: ComfyUI - Generazione audio

L'architettura esaminata prevedeva l'impiego del nodo F5 TTS Engine per la selezione e il download automatico dei modelli linguistici, mentre il nodo TTS Text per l'elaborazione del segnale acustico basato su un file audio di riferimento (utilizzato come target per il timbro), sul testo scritto e sul modello di fonetizzazione.

All'interno del nodo Character Voice è necessario inserire il testo trascritto dell'audio di riferimento per consentire un corretto campionamento, mentre il modulo Phoneme Text Normalizer si è occupato della regolarizzazione sintattica per ottimizzare la pronuncia.

Dato che i risultati qualitativi non sono stati ritenuti soddisfacenti a causa di una marcata artificialità della voce e della presenza di numerosi artefatti di quantizzazione acustica, quindi questo metodo è stato infine scartato per la produzione del video finale della tesi. Inoltre, nonostante l'integrazione di modelli compatibili con la lingua italiana, la pronuncia è risultata spesso influenzata dalle basi di addestramento fonetico in inglese.

6.5.5 Sintesi vocale: ElevenLabs

Per superare i limiti di Wan 2.2 nella gestione dell'audio, la traccia sonora è stata integrata direttamente in fase di post-produzione. Mentre nel pro-

getto originale la voce era basata sulla registrazione live di un attore, in questa sede si è scelto di affidarsi alla piattaforma di sintesi vocale basata su intelligenza artificiale ElevenLabs. Questa soluzione ha permesso di generare una voce sintetica altamente realistica e personalizzabile a partire dal testo scritto contenente le battute del personaggio, impostando la localizzazione in lingua italiana. Per garantire l'omogeneità dello stile vocale ed evitare fluttuazioni timbriche tra le diverse scene, è stato generato un unico file audio globale, successivamente segmentato in tracce minori e sincronizzato ai singoli video tramite il software Clipchamp.

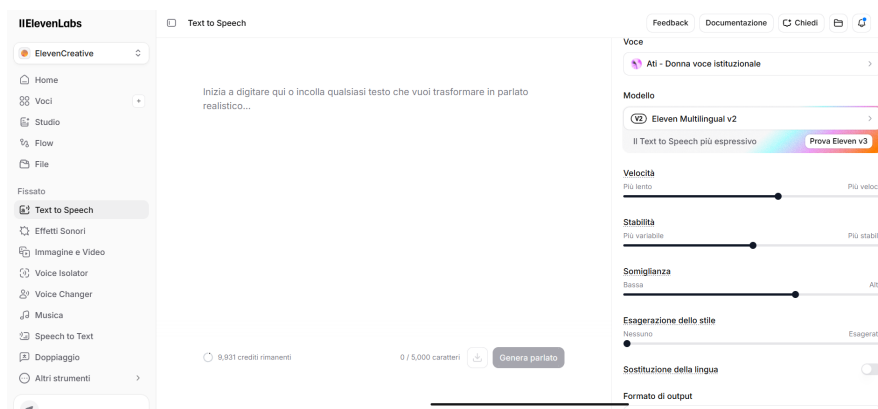


Figura 6.20: ElevenLabs - Sintesi vocale

Il controllo della generazione acustica è stato governato attraverso la calibrazione di quattro parametri fondamentali:

- **Velocità:** regola il ritmo del parlato. Valori elevati accelerano la dizione, mentre valori bassi estendono la durata delle sillabe.
- **Stabilità:** incrementare questo valore assicura una forte coerenza timbrica tra le diverse generazioni. Tuttavia, un'impostazione eccessivamente alta rischia di rendere l'intonazione monotona e artificiale. Per stringhe di testo lunghe si è preferito mantenere un valore bilanciato verso il basso per preservare le inflessioni naturali.
- **Somiglianza:** determina il grado di aderenza della voce generata rispetto al modello acustico di riferimento. Valori troppo spinti

possono introdurre artefatti metallici, è necessario effettuare varie prove per localizzare il punto di massima nitidezza.

- **Esagerazione dello stile:** amplifica l'espressività e l'enfasi drammatica del parlato. È un parametro estremamente sensibile (impostato di default a 0) in cui anche minimi incrementi possono compromettere la corretta fonetica italiana o introdurre asimmetrie nella pronuncia.

6.6 Valutazione dei risultati

Sono stati numerosi i test di generazione, alcuni di essi hanno portato ad artefatti evidenti. Ad esempio, oggetti di sfondo che si muovevano o materializzavano deformati in mano al personaggio e movimenti troppo bruschi o imprevisti. Tuttavia, con un processo di fine-tuning del prompt e di tentativi variando il seme di partenza, si sono ottenuti ottimi risultati.

Una volta completata la raffinazione dei singoli segmenti visivi tramite Wan 2.2, la correzione fotogramma per fotogramma degli ultimi frame critici in ambiente locale (mediante i nodi Expression Editor) e la scomposizione della traccia vocale sintetica creata su ElevenLabs, i materiali sono stati montati da un montatore professionista producendo così il video definitivo e completo.

Il prodotto multimediale finale si configura come un video della durata complessiva di 1 minuto e 29 secondi, incentrato sulla figura dell'asset digitale di Ati all'interno della ricostruzione tridimensionale della tomba etrusca di Cerveteri. Nel filmato, il personaggio interagisce direttamente con lo spettatore mostrando un ventaglio rituale dell'epoca e illustrandone il significato storico e simbolico.

L'analisi del filmato finale permette di confermare il raggiungimento degli obiettivi metodologici prefissati in apertura di capitolo:

- **Costanza del personaggio:** l'adozione di Wan 2.2 in precisione FP8 Scaled, supportata dall'azione combinata dei LoRA di accelerazione e dei moduli di campionamento avanzati, ha garantito

una stabilità dei tratti somatici e vestimentari di Ati impossibile da ottenere con hardware consumer.

- **Fluidità:** i micro-movimenti del volto e del personaggio risultano naturali e privi di sfarfallii e artefatti geometrici.
- **Integrazione audio:** nonostante i limiti nativi dell'architettura DiT nel calcolo dell'audio, la tecnica di correzione manuale degli ultimi frame combinata al montaggio della traccia di ElevenLabs ha permesso di ottenere una sincronizzazione labiale (Lip-Sync) e un'espressività generale coerenti con gli standard qualitativi richiesti dal progetto.

Il video così ottenuto rappresenta la sintesi tangibile della tesi, dimostrando l'efficacia dell'integrazione tra metodologie di computazione ad alte prestazioni e flussi di lavoro basati su intelligenza artificiale generativa per la valorizzazione del patrimonio culturale.

Conclusioni

La presente ricerca ha tracciato un percorso lineare, rigoroso e sistematico nell'ambito dell'applicazione delle tecnologie di Intelligenza Artificiale Generativa *open-source* e *open-weight* orientate alla valorizzazione del patrimonio culturale. Il lavoro svolto ha dimostrato concretamente come l'evoluzione dei modelli di diffusione e l'avvento delle architetture basate su Transformer possano ridefinire gli standard metodologici della museologia digitale, riuscendo a coniugare un'elevata espressività visivo-cinematografica con l'accuratezza storica e filologica.

Il percorso sperimentale ha evidenziato un netto divario metodologico e tecnologico tra l'approccio locale basato su hardware di livello consumer e la scalabilità computazionale offerta dall'infrastruttura di High-Performance Computing (HPC).

Nelle fasi di prototipazione iniziale, l'addestramento condotto sul modello Stable Diffusion 1.5 tramite l'interfaccia Kohya_ss ha messo in luce severi colli di bottiglia strutturali, legati principalmente ai limiti di memoria VRAM e alla risoluzione nativa di soli 512×512 pixel. Tali vincoli hanno reso necessaria l'applicazione di complesse tecniche di ottimizzazione, sia nella gestione dei parametri di training sia nella fase di strutturazione e pulizia del dataset.

I successivi test effettuati in configurazione Multi-LoRA hanno confermato queste criticità, mostrando evidenti fenomeni di saturazione latente e fenomeni di interferenza semantica a livello delle matrici di Cross-Attention. In presenza di scenari ricchi di dettagli e moduli multirango sovrapposti, la rete U-Net convoluzionale si è dimostrata incapace

di gestire simultaneamente l'allineamento di prompt testuali complessi senza generare allucinazioni visive o una marcata degradazione stilistica e geometrica. Queste limitazioni, teorizzate in fase preliminare e confermate empiricamente al termine dei test locali, hanno decretato l'indispensabilità di una transizione verso un'infrastruttura di calcolo massiva.

La migrazione sul supercomputer Leonardo (CINECA) ha permesso di superare radicalmente i limiti dell'ambiente locale, sebbene abbia introdotto sfide ingegneristiche non banali. La complessità principale risiedeva nella necessità di mappare e riconfigurare i workflow di addestramento standard all'interno di un ambiente di calcolo HPC che opera rigorosamente offline a runtime. Attraverso un'analisi approfondita degli script di esecuzione e delle dipendenze dei comandi di addestramento, sono state apportate modifiche mirate al codice per consentire l'esecuzione dei processi in totale assenza di connessione alla rete esterna.

Parallelamente, per soddisfare l'esigenza di abbattere i tempi di training, l'architettura d'addestramento è stata scalata per sfruttare appieno l'approccio distribuito multi-GPU. Mediante l'utilizzo di Chappyner per operare più agevolmente sul nodo di calcolo di Leonardo e l'adozione del framework avanzato musubi-tuner, la ricerca ha sfruttato la potenza computazionale dei nodi della partizione Booster di Leonardo, equipaggiati con GPU NVIDIA A100 da 64 GB. Questo cambio di scala ha abilitato il fine-tuning distribuito su modelli a parametri massivi, consentendo la gestione offline di pesanti architetture di condizionamento testuale basate sul modello T5-XXL.

Il fulcro scientifico della tesi si è concentrato sulla ricerca di un delicato punto di equilibrio tra la flessibilità generativa dello strumento sintetico e il rigore filologico richiesto come obiettivo. In questo contesto, l'analisi comparativa dei modelli DiT ha decretato il definitivo superamento dei precedenti approcci puramente convoluzionali, grazie all'analisi di modelli più complessi e potenti:

- **Flux.1 e Z-Image:** entrambi i modelli hanno introdotto una su-

periore coerenza geometrica globale rispetto alle tecnologie passate, un risultato diretto del calcolo delle relazioni d'attenzione a lungo raggio che ha permesso di preservare accuratamente le simmetrie dei manufatti e delle strutture architettoniche complesse, quali il Tempio e la Tomba Etrusca. Tuttavia, l'analisi ha rivelato che Flux.1, pur dimostrandosi capace di restituire ottime generazioni in presenza di soggetti singoli, mostra limiti qualitativi e di coerenza geometrica in caso di configurazioni Multi-LoRA. Per quanto riguarda Z-Image, il modello ha offerto buoni risultati in fase di generazione locale grazie alla sua ottimizzazione, ma il suo addestramento è risultato computazionalmente proibitivo su hardware consumer. Il training eseguito su Leonardo ha dimostrato l'efficacia di Z-Image in configurazioni semplici, seppur con minime imprecisioni. Tuttavia, di fronte a scenari Multi-LoRA, il modello ha manifestato una forte contaminazione di stile e geometria tra i diversi soggetti in scena, un fenomeno di bleeding che ne ha precluso l'utilizzo per gli standard richiesti.

- **Flux.2 [dev]:** forte dei suoi 32 miliardi di parametri e dell'integrazione nativa del Vision-Language Model Mistral-3, questo modello si è imposto come la soluzione più performante e robusta dell'intera ricerca. L'architettura Asymmetric MMDiT-2 e l'algoritmo di Flow Matching di seconda generazione hanno drasticamente azzerato i fenomeni di prompt dropping, garantendo una generazione Ultra-High-Definition nativa. Questa capacità ha permesso di mappare texture microscopiche direttamente in fase di prima inferenza, escludendo l'adozione di moduli di upscaling esterni che avrebbero rischiato di alterare o inventare i dettagli storici originali. Trattandosi di un modello molto grande, Flux.2 ha richiesto specifiche ottimizzazioni per prevenire errori di saturazione della memoria (Out-Of-Memory) anche sulle architetture HPC. La sperimentazione ha dimostrato che il dimezzamento della precisione del modello durante il training ha consentito l'ottimizzazione delle

risorse senza comportare alcuna perdita avvertibile nella qualità del prodotto visivo finale. Di conseguenza, Flux.2 ha rappresentato la vera colonna portante dell'intera fase di generazione d'immagine, dimostrando una costanza e una precisione nettamente superiori alla concorrenza sia sui soggetti singoli sia nelle complesse interazioni Multi-LoRA.

La fase conclusiva della pipeline di ricerca ha affrontato la transizione alla dimensione temporale all'interno del Capitolo 5, focalizzandosi sulla sperimentazione comparativa tra i modelli video open-weights Wan 2.2 e LTX 2, testati inizialmente in locale e successivamente scalati sull'infrastruttura HPC.

La sperimentazione in locale, condotta attraverso l'orchestratore Pinokio e l'applicazione Wan2GP, ha permesso di verificare le potenzialità dei modelli su hardware limitati (8 GB di VRAM). I risultati di questa fase sono tuttavia emersi come insufficienti sotto il profilo della qualità puramente visiva e dell'accuratezza filologica. La necessità di ricorrere a modelli fortemente quantizzati per consentire l'esecuzione dei processi ha degradato l'output, richiedendo peraltro tempi di elaborazione estremamente dilatati. In questo scenario locale, LTX 2 e Wan 2.2 hanno mostrato performance qualitative sovrapponibili, sebbene LTX 2 presentasse il grande vantaggio di includere una traccia audio nativa.

Il pieno sfruttamento delle potenzialità di tali modelli ha richiesto nuovamente l'intervento delle risorse di Leonardo. Sviluppando appositi workflow all'interno dell'ambiente ComfyUI, è stato possibile generare output video ad alta definizione sfruttando le varianti a massima precisione di entrambi i software:

- **LTX 2:** pur dimostrandosi completo e cinematicamente preciso nella gestione dei movimenti di macchina, ha manifestato una tendenza eccessiva al fotorealismo, portando a sistematiche variazioni geometriche di Ati. Tale deriva stilistica ha spinto a escludere il modello dalla produzione finale.

- **Wan 2.2:** ha offerto una risposta decisamente più rigorosa dal punto di vista filologico, supportata da una qualità visiva complessiva superiore. L'adozione di tecniche avanzate di controllo cinematografico, la manipolazione diretta dello spazio latente e l'applicazione di moduli per l'espressione facciale (mediante Live Portrait) hanno permesso la sintesi di un video della durata di circa un minuto e mezzo.

L'unico svantaggio riscontrato nella selezione di Wan 2.2 è risieduto nella mancanza nativa del supporto audio. Per colmare questa lacuna, sono stati esplorati diversi tentativi direttamente in ComfyUI.

Il primo ha previsto l'implementazione di un workflow basato sul nodo WanSoundImageToVideo, concepito per processare in modo sincrono l'immagine di riferimento e una traccia acustica guida, il test ha tuttavia restituito una qualità video finale nettamente inferiore rispetto alle generazioni pure, costringendo a scartare l'opzione. Un secondo tentativo basato su un workflow per la generazione isolata di una traccia audio da montare in post-produzione ha restituito risultati qualitativamente non accettabili per gli standard del progetto.

La risoluzione del problema è stata ottenuta attraverso l'utilizzo della piattaforma di intelligenza artificiale ElevenLabs. La clonazione e la generazione di una voce sintetica dedicata hanno permesso di strutturare il parlato tramite algoritmi di Text-To-Speech di ultima generazione, garantendo un accompagnamento narrativo profondo e realistico.

L'intero flusso di lavoro, così ottimizzato e stratificato, è culminato nella presentazione ufficiale dei contenuti digitali in occasione della mostra immersiva "Experience Etruria", ospitata presso l'Istituto Italiano di Cultura di San Francisco nel maggio 2026, decretando il risultato pratico del framework proposto.

Sviluppi futuri

Nonostante i traguardi raggiunti, la ricerca ha delineato alcuni limiti tecnologici che aprono la strada a importanti evoluzioni future:

- **Ottimizzazione del bleeding materico nei sistemi DiT:** sebbene l'architettura basata su Flux.2 [dev] abbia sensibilmente ridotto gli artefatti visivi rispetto ai precedenti approcci convoluzionali, l'interazione simultanea di moduli Multi-LoRA sulle reti Transformer presenta ancora rari fenomeni di bleeding. Questa criticità si presenta quando si combinano simultaneamente un personaggio e un oggetto archeologico. Un possibile sviluppo futuro è quello di esplorare algoritmi avanzati di fusione asimmetrica dei pesi delle matrici a runtime, oppure verso l'ingegnerizzazione e l'addestramento di LoRA gerarchici (Hierarchical LoRA), capaci di isolare rigorosamente i vettori di stile da quelli puramente geometrico-strutturali.
- **Transizione nativa al Text-to-3D filologico:** l'attuale workflow operativo si affida a un approccio ibrido, che prevede il rendering bidimensionale di modelli poligonali preventivamente ricostruiti in Blender per istruire i successivi modelli generativi 2D o i motori di animazione video. Il naturale e necessario passo successivo risiede nell'integrazione di pipeline generative Text-to-3D e Image-to-3D di ultima generazione. L'adozione di tecnologie emergenti basate su Campi di Radianza Neurale (NeRF) o sul 3D Gaussian Splatting permetterebbe di superare la sintesi bidimensiona-

le intermediaria, generando direttamente geometrie tridimensionali originarie vincolate da metriche e restauri archeologici rigorosi.

- **Scalabilità del framework ed estensione del catalogo espositivo:** il flusso di lavoro formalizzato in questa tesi ha dimostrato una spiccata predisposizione alla modularità e alla replicabilità sistematica. Una linea di sviluppo immediata riguarda l’espansione del database dei soggetti storici gestiti. Grazie alla standardizzazione delle procedure di training individuate, l’inclusione di nuovi reperti o figure storiche richiede esclusivamente l’addestramento di un LoRA specifico. Una volta integrato nel sistema, la combinazione dei pesi e la generazione di immagini all’interno di scenari contestuali inediti si rivela un’operazione semplice, scalabile e precisa, aprendo alla possibilità di digitalizzare numerosi elementi di cataloghi museali.
- **Esplorazione sistematica e benchmarking di modelli emergenti:** il panorama dell’Intelligenza Artificiale applicata alla sintesi multimediale si muove a un ritmo evolutivo estremamente rapido, con soluzioni open-source costantemente rilasciate dalla community scientifica. Una direttrice futura fondamentale consiste nel monitoraggio e nell’integrazione di nuove versioni dei modelli già testati o di architetture del tutto inedite. Sarà prioritario istituire un protocollo di benchmarking quantitativo e qualitativo per valutare se i futuri modelli mantengano o superino il rigore filologico qui conseguito, ottimizzando al contempo l’impatto computazionale e le richieste di memoria latente.
- **Ingegnerizzazione e ottimizzazione di nuovi workflow in ambiente ComfyUI:** nel corso dell’attività di ricerca sono stati mappati e testati numerosi flussi di lavoro nodali, ma la flessibilità intrinseca di ComfyUI consente una costante evoluzione strutturale. Uno sviluppo futuro di grande rilievo pratico prevede la continua sperimentazione e il fine-tuning dell’architettura dei no-

di, attingendo alle ultime soluzioni create dagli sviluppatori della community. Un'espansione di quanto già visto è la possibilità di implementare workflow specializzati nella risoluzione di svantaggi ben noti dei modelli, ottimizzando l'orchestrazione dei parametri per massimizzare l'efficienza complessiva senza saturare le risorse hardware.

Glossario

Bleeding: fenomeno indesiderato in cui caratteristiche visive, stili o texture di un soggetto contaminano involontariamente un altro elemento presente nella stessa scena.

Checkpoint: file che racchiude l'intero stato dei pesi pre-addestrati di una rete neurale a un determinato stadio dello sviluppo. Rappresenta il modello base (come Flux.2 [dev]) da cui partire per la generazione o per successivi addestramenti.

HPC (High-Performance Computing): l'utilizzo di supercomputer e sistemi di calcolo parallelo ad altissime prestazioni per eseguire elaborazioni computazionali massive che richiederebbero tempi proibitivi su hardware consumer. Nel contesto della presente ricerca, fa specifico riferimento all'infrastruttura del supercomputer Leonardo (CINECA) impiegata per l'orchestrazione del training distribuito multi-GPU.

Inferenza: la fase operativa in cui un modello di Intelligenza Artificiale già addestrato elabora l'input fornito dall'utente per generare un output (l'immagine sintetica o il video), senza modificare i propri pesi interni.

Inpainting: tecnica di generazione mirata che permette di modificare, correggere o ricostruire solo una specifica porzione delimitata di un'immagine tramite una maschera e un prompt testuale, preservando la coerenza del contesto circostante.

LoRA (Low-Rank Adaptation): tecnica di fine-tuning efficiente che permette di specializzare un modello inserendo un numero ridotto di parametri aggiuntivi (matrici a basso rango) senza dover modificare i pesi originali della rete, ideale per introdurre nuovi soggetti o stili specifici.

Prompt: stringa di testo in linguaggio naturale fornita come input a un modello generativo per descrivere, indirizzare e guidare le caratteristiche semantiche, stilistiche e strutturali dell'output desiderato.

Quantizzazione: tecnica di ottimizzazione computazionale che riduce la precisione numerica dei pesi di un modello per diminuire drasticamente l'occupazione di memoria VRAM e velocizzare i tempi di calcolo, accettando un minimo compromesso sulla qualità finale.

Token: l'unità fondamentale in cui un testo viene frammentato dal Tokenizer affinché possa essere elaborato matematicamente da un modello linguistico. Può corrispondere a una parola intera, a una sillaba o a un singolo carattere.

VRAM (Video Random Access Memory): memoria ad accesso casuale ad altissima velocità integrata direttamente sulle unità di elaborazione grafica (GPU). Costituisce la risorsa hardware più critica e il principale collo di bottiglia durante l'addestramento e l'inferenza di modelli generativi a parametri massivi, determinando la massima risoluzione e complessità architetturale gestibile.

Appendice A

Codice Particolarmente significativo

La presente appendice racchiude parti importanti del codice di *flux2_utils.py* in modo da mostrarne le modifiche effettuate, i file dataset e dei comandi in formato .TOML e lo script python lanciato in Blender per garantire l'animazione fisica corretta delle trecce del personaggio Ati.

Script *flux2_utils.py*, gestione delle dipendenze offline per HPC Leonardo:

```
1 import argparse
2 import json
3 import numpy as np
4 import torch
5 import math
6 from dataclasses import dataclass
7 from accelerate import init_empty_weights
8 from einops import rearrange
9 from typing import Optional, Tuple, Union
10 from torch import Tensor, nn
11 from transformers import (
12     Mistral3ForConditionalGeneration,
13     Mistral3Config,
14     AutoProcessor,
15     Qwen2Tokenizer,
16     Qwen3ForCausalLM,
17 )
18
19 # Integrazione con l'infrastruttura musubi-tuner per HPC
20 from musubi_tuner.dataset.image_video_dataset import (
21     ARCHITECTURE_FLUX_2_DEV,
22     ARCHITECTURE_FLUX_2_DEV_FULL,
23     BucketSelector,
24 )
25 from musubi_tuner.modules.fp8_optimization_utils import
    apply_fp8_monkey_patch
```

```

26 from musubi_tuner.utils.lora_utils import
    load_safetensors_with_lora_and_fp8
27
28 # Path assoluto all'interno dello scratch di Leonardo per l'
    esecuzione offline
29 M3_TOKENIZER_ID = "/leonardo_scratch/large/userexternal/mmancin2/
    Kohya_install/musubi-tuner/mistralai/Mistral-Small-3.1-24B-
    Instruct-2503"
30 OUTPUT_LAYERS_MISTRAL = [10, 20, 30]
31 MAX_LENGTH = 512
32 SYSTEM_MESSAGE = """You are an AI that reasons about image
    descriptions. You give structured responses focusing on object
    relationships, object attribution and actions without
    speculation."""
33
34 @dataclass(frozen=True)
35 class Flux2ModelInfo:
36     params: Flux2Params
37     defaults: dict[str, float | int]
38     fixed_params: set[str]
39     guidance_distilled: bool
40     architecture: str
41     architecture_full: str
42     qwen_variant: Optional[str] = None # None per Mistral

```

Listing A.1: Inizializzazione di flux2_utils ed esecuzione offline

Il seguente frammento mostra il metodo *forward* patchato per isolare dinamicamente il tokenizer di Mistral-3 e forzare la lunghezza del contesto a 256 token:

```

1 def forward(self, txt: list[str]):
2     if not isinstance(txt, list):
3         txt = [txt]
4
5     # Gestione dinamica del Processor/Tokenizer per Mistral-3
6     tokenizer_to_use = self.tokenizer.tokenizer if hasattr(self,
    tokenizer, "tokenizer") else self.tokenizer
7
8     inputs = tokenizer_to_use(
9         txt,
10         padding="max_length",
11         max_length=256, # Fix architetturale imposto per la
    saturazione a 256 token
12         truncation=True,
13         return_tensors="pt",
14     ).to(self.device)
15
16     # Trasferimento dei tensori alla memoria VRAM della GPU A100

```

```

17     input_ids = inputs["input_ids"].to(self.mistral3.device)
18     attention_mask = inputs["attention_mask"].to(self.mistral3.
19 device)
20
21     # Passaggio di forward con estrazione degli stati nascosti (
22 hidden states) senza caching
23     output = self.mistral3(
24         input_ids=input_ids,
25         attention_mask=attention_mask,
26         output_hidden_states=True,
27         use_cache=False,
28     )

```

Listing A.2: Logica di forward pass ed estrazione delle feature testuali da Mistral-3

Script Python lanciato in Blender *atitrecce_bake_dinamic*:

```

1  import bpy
2  import os
3
4
5  def setup_cloth(obj, frame_start, frame_end, pre_roll):
6      cloth_mod = obj.modifiers['Cloth']
7      #cloth_mod.render = True
8      #cloth_mod.realtime = True
9
10     # cloth_mod.settings.pre_roll = pre_roll # RIMOSSO: Non esiste
11     piu in Blender 3.x
12
13     point_cache = cloth_mod.point_cache
14     point_cache.frame_start = frame_start - pre_roll # non si puo
15     fare da GUI
16     print("point_cache.frame_start " + str(point_cache.frame_start)
17 )
18
19     point_cache.frame_end = frame_end
20     point_cache.frame_step = 1
21     point_cache.use_disk_cache = True
22
23
24  def bake_trecce_with_preroll():
25     base_dir, base_name = os.path.split(bpy.data.filepath)
26     base_name = os.path.splitext(base_name)[0]
27     frame_start = bpy.context.scene.frame_start
28     frame_end = bpy.context.scene.frame_end
29     pre_roll = 0 #Questo parametro deve essere uguale al preroll
30     impostato nelle proprieta' GUI del cloth

```

```

27     # Modifica per Blender 2.8+ / 3.x: user_preferences ->
preferences
28     if hasattr(bpy.context, "preferences"):
29         bpy.context.preferences.edit.use_negative_frames = True
30     else:
31         bpy.context.user_preferences.edit.use_negative_frames =
True
32     scenes_to_bake = ['Scene', 'comp', 'solid', 'wire']
33
34     for scene_name in scenes_to_bake:
35         try:
36             sce = bpy.data.scenes[scene_name]
37             # Imposta la scena corrente per assicurare che il
contesto sia corretto
38             bpy.context.window.scene = sce
39
40             # Spegni la collection "Collection 11" nel View Layer
corrente
41             collection_name = "Collection 11"
42             vl = bpy.context.view_layer
43             if collection_name in vl.layer_collection.children:
44                 vl.layer_collection.children[collection_name].
exclude = True
45
46             sce.use_gravity = True
47             sce.frame_current = - pre_roll
48
49             bpy.ops.ptcache.free_bake_all()
50
51             obj1 = bpy.data.objects["TrecceFronteCloth"]
52             setup_cloth(obj1, frame_start, frame_end, pre_roll)
53
54             obj2 = bpy.data.objects["TrecceClothNew"]
55             setup_cloth(obj2, frame_start, frame_end, pre_roll)
56
57             obj1 = bpy.data.objects["TrecceFronteCloth.001"]
58             setup_cloth(obj1, frame_start, frame_end, pre_roll)
59
60             obj2 = bpy.data.objects["TrecceClothNew.001"]
61             setup_cloth(obj2, frame_start, frame_end, pre_roll)
62
63             print(f"bake begin for scene {scene_name}")
64             bpy.ops.ptcache.bake_all(bake=True)
65             print("done")
66         except KeyError:
67             print(f"Scene '{scene_name}' not found, skipping.")
68         except Exception as e:
69             print(f"Error baking scene '{scene_name}': {e}")
70

```

```
71 bake_trecce_with_preroll()
```

Listing A.3: atitrecce_bake_dinamic - correttezza fisica delle trecce durante l'animazione in Blender

File TOML *dataset_Flux2_Ati.toml* di dataset per il training con Flux.2 per Ati:

```
1 [general]
2 resolution = [1024, 1024]
3 caption_extension = ".txt"
4 batch_size = 1
5 enable_bucket = true
6
7 [[datasets]]
8 image_directory = "/leonardo_scratch/large/userexternal/mmancin2/
   Kohya_install/dataset/AtiFlux2/AtiFlux2"
9 num_repeats = 4
```

Listing A.4: File TOML di dataset per Ati - Flux.2

Bibliografia e Sitografia

- [1] Hearneman AI. Train an ai influencer lora with z image turbo on runpod. <https://www.youtube.com/watch?v=Kf0TWLcagow>, 2026.
- [2] Black Forest Labs. Flux.2-dev model repository on hugging face. <https://huggingface.co/black-forest-labs/FLUX.2-dev>, 2026.
- [3] Black Forest Labs. Flux.2 github repository. <https://github.com/black-forest-labs/flux2>, 2026.
- [4] Black Forest Labs. Flux.2 model family. <https://bfl.ai/models/flux-2>, 2026.
- [5] Laura Carnevali. How to train your own lora - stable diffusion fine-tuning, 2023. URL: <https://www.youtube.com/watch?v=xholR62Q2tY>.
- [6] CINECA. Leonardo HPC system documentation, 2024. URL: <https://leonardo-supercomputer.cineca.eu/it/leonardo-hpc-system/>.
- [7] CINECA Interactive Computing. Chappyner: Orchestration and deployment of interactive web services on HPC systems, 2024. URL: https://gitlab.hpc.cineca.it/interactive_computing/chappyner/-/tree/main.
- [8] ComfyOrg. Comfyui documentation - basicscheduler. <https://docs.comfy.org/built-in-nodes/BasicScheduler>, 2026.
- [9] ComfyOrg. Comfyui documentation - ksampler node. <https://docs.comfy.org/built-in-nodes/sampling/ksampler>, 2026.
- [10] ComfyOrg. Comfyui official github organization. <https://github.com/Comfy-0rg/>, 2026.

- [11] Comfyui ltx 2 video generation tutorial. URL: <https://docs.comfy.org/tutorials/video/ltx/ltx-2>.
- [12] Comfyui wan 2.2 integration guide. URL: https://docs.comfy.org/tutorials/video/wan/wan2_2.
- [13] Comfyui wan 2.1 funcontrolvideo workflow. URL: <https://docs.comfy.org/tutorials/video/wan/fun-control>.
- [14] CompVis. Latent diffusion models github repository. <https://github.com/CompVis/latent-diffusion>, 2022.
- [15] d4N87. Guida training lora flux con fluxgym. <https://www.youtube.com/watch?v=FesqxPBpaRU>, 2024.
- [16] Shakir Mohamed Danilo Jimenez Rezende. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015. URL: <https://arxiv.org/abs/1505.05770>.
- [17] deepbeepmeep. Wan2gp open-source infrastructure. URL: <https://github.com/deepbeepmeep/Wan2GP>.
- [18] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021. URL: <https://arxiv.org/pdf/2110.02861>.
- [19] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Mangold, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. *arXiv preprint arXiv:2403.03206*, 2024. URL: <https://arxiv.org/abs/2403.03206>.
- [20] GeeksforGeeks. Distributed data parallel. <https://www.geeksforgeeks.org/deep-learning/distributed-data-parallel/>, 2025.
- [21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *arXiv preprint arXiv:1406.2661*, 2014. URL: <https://arxiv.org/abs/1406.2661>.

- [22] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. URL: <https://arxiv.org/pdf/1706.02677>.
- [23] Or Greenberg. Demystifying flux architecture. *arXiv preprint arXiv:2507.09595*, 2025. URL: <https://arxiv.org/abs/2507.09595>.
- [24] Yoav HaCohen, Benny Brazowski, Nisan Chiprut, Yaki Bitterman, Andrew Kvochko, Avishai Berkowitz, Daniel Shalem, Daphna Lifschitz, Dudu Moshe, Eitan Porat, Eitan Richardson, Guy Shiran, Itay Chachy, Jonathan Chetboun, Michael Finkelson, Michael Kupchick, Nir Zabari, Nitzan Guetta, Noa Kotler, Ofir Bibi, Ori Gordon, Poriya Panet, Roi Benita, Shahar Armon, Victor Kulikov, Yaron Inger, Yonatan Shiftan, Zeev Melumian, and Zeev Farbman. Ltx-2: Efficient joint audio-visual foundation model. *arXiv preprint arXiv:2601.03233*, 2026. URL: <https://arxiv.org/pdf/2601.03233>.
- [25] IBM Think. Variational autoencoder (VAE). <https://www.ibm.com/it-it/think/topics/variational-autoencoder>, 2024.
- [26] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. URL: <https://arxiv.org/abs/1312.6114>.
- [27] Kohya_ss. Musubi-tuner: Advanced fine-tuning framework for large-scale transformer models, 2024. URL: <https://github.com/kohya-ss/musubi-tuner>.
- [28] Kohya_ss. Musubi-tuner documentation - dataset configuration guide. https://github.com/kohya-ss/musubi-tuner/blob/main/docs/dataset_config.md, 2024.
- [29] Bernard Maltais. kohya_ss: Gui for kohya’s stable diffusion scripts, 2024. URL: https://github.com/bmaltais/kohya_ss.
- [30] Sharan Narang, Gregory Diamos, and Erich Elsen. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. URL: <https://arxiv.org/pdf/1710.03740>.

- [31] An Nyantudre. Florence-2-vision-language-model implementation repository, 2024. URL: <https://github.com/anyantudre/Florence-2-Vision-Language-Model>.
- [32] OstrisAI. How to train a flux.2 lora with ai toolkit. <https://www.youtube.com/watch?v=qWDpPos6vrI>, 2026.
- [33] Pinokio - localhost platform for humans and ai. URL: <https://desktop.pinokio.co/>.
- [34] PyTorch Documentation. Torch.optim.adamw reference manual, 2024. URL: <https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html>.
- [35] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, and Gretchen Krueger. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021. URL: <https://arxiv.org/abs/2103.00020>.
- [36] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *arXiv preprint arXiv:2112.10752*, 2022. URL: <https://arxiv.org/pdf/2112.10752>.
- [37] r/StableDiffusion. Can you use several loras simultaneously? how do they work? one after the other? https://www.reddit.com/r/StableDiffusion/comments/18dhvep/can_you_use_several_loras_simultaneously_how_do/, 2023.
- [38] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. *arXiv preprint arXiv:2208.12242*, 2022. URL: <https://arxiv.org/pdf/2208.12242>.
- [39] RunComfy. Comfyui nodes: Sampler-scheduler settings (jps). https://www.runcomfy.com/comfyui-nodes/ComfyUI_JPS-Nodes/Sampler-Scheduler-Settings--JPS-, 2026.
- [40] SECourses. Training sdxl with lora - full guide, 2024. URL: <https://www.youtube.com/watch?v=AY6DMBCIZ3A>.

- [41] SECourses. Flux lora training simplified: From zero to hero with kohya ss gui (8gb gpu, windows) tutorial guide. <https://www.youtube.com/watch?v=nySGu12Y05k>, 2025.
- [42] Stability AI. Generative models github repository. <https://github.com/Stability-AI/generative-models>, 2023.
- [43] Stable Diffusion Art. How to use lora (low-rank adaptation) in stable diffusion. <https://stable-diffusion-art.com/lora/>, 2023.
- [44] Stable Diffusion Art. How to use vae in stable diffusion. <https://stable-diffusion-art.com/how-to-use-vae/>, 2024.
- [45] Stable Diffusion Art. Stable diffusion samplers: A complete guide. <https://stable-diffusion-art.com/samplers/>, 2024.
- [46] Tongyi-MAI. Z-image: Official implementation and model weights. <https://github.com/Tongyi-MAI/Z-Image>, 2025.
- [47] Differenze wan 2.1 e 2.2. URL: <https://www.youtube.com/watch?v=Fs-gKrpm9nc&t=12s>.
- [48] Wan Team, Alibaba Group. Wan: open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. URL: <https://arxiv.org/pdf/2503.20314>.
- [49] Lilian Weng. Flow-based generative models. <https://lilianweng.github.io/posts/2018-10-13-flow-models/>, 2018.
- [50] Lilian Weng. From autoencoder to beta-VAE. <https://lilianweng.github.io/posts/2018-08-12-vae/>, 2018.
- [51] Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks. *arXiv preprint arXiv:2311.06242*, 2023. URL: <https://arxiv.org/pdf/2311.06242>.
- [52] Alibaba Group Z-Image Team. Z-image: An efficient image generation foundation model with single-stream diffusion transformer. *arXiv preprint arXiv:2511.22699*, 2025. URL: <https://arxiv.org/abs/2511.22699>.

Ringraziamenti

Desidero esprimere la mia più profonda gratitudine alla mia relatrice, la Professoressa Serena Morigi per la sua disponibilità, la sua guida e il tempo che mi ha dedicato.

Un sincero ringraziamento va a tutti lavoratori del CINECA, con cui ho avuto la preziosa opportunità di interfacciarmi e che mi ha permesso di inserirmi in un contesto stimolante, caratterizzato da un costante scambio di idee e da una forte collaborazione. In particolare, desidero ringraziare l'ingegnere Antonella Guidazzoli, responsabile del progetto, per avermi accolto e supportato durante l'intera esperienza. Un ringraziamento speciale va a Daniele, Alex, Federico, Gabriele e Giorgia, che mi hanno fatto sentire da subito parte integrante del gruppo e si sono dimostrati sempre disponibili in ogni momento di necessità. Più in generale, grazie a tutto lo staff CINECA: il periodo trascorso in azienda è stato estremamente prezioso, sia a livello professionale che umano.

Un ringraziamento speciale va ai miei genitori, per l'amore che mi avete trasmesso in tutti questi anni lontano da casa e per i sacrifici fatti per permettermi di raggiungere i miei obiettivi. Grazie per il sostegno e la fiducia che avete riposto in me, spesso più di quanta io ne avessi in me stesso. Ogni volta che ho dubitato di me, voi siete stati sempre pronti ad ascoltarmi, capirmi e darmi nuova motivazione e sicurezza. Siete stati fantastici e sono felice della persona che sono diventato, grazie a voi e all'esempio che mi avete dato.

A Giuseppe, mio fratello, il mio punto di riferimento quando guardo

al futuro con incertezza. Grazie per avermi sempre fatto sentire il tuo supporto, il tuo amore e per aver tracciato il percorso andando in avanscoperta, permettendomi di affrontare le sfide con maggiore sicurezza. I tuoi consigli sono stati fondamentali in molti momenti. Non ci sono parole per esprimere quanto ti voglio bene, sei il miglior fratello maggiore che potessi chiedere. Grazie anche a Maria e la piccola Marta, sono davvero felice di avervi nella mia vita e di poter condividere con voi questi momenti di gioia.

A Roberta. Sei il mio raggio di sole, la mia forza e la mia quotidianità. Non c'è stato giorno in cui tu non mi abbia sostenuto, incoraggiato e aiutato. Non abbiamo bisogno delle parole per capirci (ehy!) e questa è una cosa stupenda. La tua presenza in questo percorso è stata fondamentale, grazie per tutto ciò che mi dai.

A Laura, grazie per la tua preziosa amicizia, per i momenti di svago e di sfogo, per le risate e le chiacchierate. Ti sei sempre dimostrata una persona speciale, disponibile e comprensiva, è sempre bello passare del tempo insieme.

A Francesca, ci siamo riavvicinati da quando sei venuta a Bologna e sono felice che sia accaduto. Grazie per la tua spontaneità, la tua simpatia e per la leggerezza che mi hai trasmesso. Ora un bel passo di *Salsa* mentre mangiamo un po' di *'nduja* e siamo a posto.

A Federica e Filippo, amici e compagni in questo lungo percorso. Grazie per aver reso le giornate di studio, gli esami e le lezioni più leggere e divertenti, rendendo l'università un'esperienza migliore e completa. Ora una birra (*IPA* rigorosamente alla spina) è d'obbligo.

Infine, un grazie sincero a tutti gli amici che, in un modo o nell'altro, mi hanno supportato, incoraggiato e accompagnato lungo questo percorso.