Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Story Visualization via Masked Generative Transformers with Character Guidance and Caption Augmentation

Διπλωματικη Εργασια

του

Χρήστου Παπαδημητρίου

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης

Story Visualization via Masked Generative Transformers with Character Guidance and Caption Augmentation

Διπλωματικη Εργασια

του

Χρήστου Παπαδημητρίου

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Μαρτίου, 2024.

.....	.....	.....
Γεώργιος Στάμου	Αθανάσιος Βουλόδημος	Στέφανος Κόλλιας
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2024

.....

Χρηστος Παπαδημητριου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η Οπτικοποίση Ιστορίας (ΟΙ) είναι μία απαιτητική εργασία στην Τεχτητή Νοημοσύνη, η οποία εμπίπτει στην τομή μεταξή Επεξεργασίας Φυσικής Γλώσσας και Όρασης Υπολογιστών. Η εργασία αυτή συνίσταται στην παραγωγή μιάς ακολουθίας εικόνων που αποτελούν οπτικοποίηση μίας ακολουθίας προτάσεων. Οι προτάσεις σχηματίζουν μία συνεκτική αφήγηση και το ίδιο πρέπει να συμβαίνει με τις εικόνες. Η ΟΙ εισήχθει για πρώτη φορά το 2019[18] και απο τότε έχει αντιμετωπιστεί με διάφορες τεχνικές, συμπεριλαμβανομένου των GANs [18, 36, 20, 19], των Μετασχηματιστών (Transformers)[5, 1] και των μεθόδων Διάχυσης (Diffusion)[25, 33].

Σε αυτή την διπλωματική επιχειρούμε να προσεγγίσουμε την Οπτικοποίση Ιστορίας βασισμένοι σε μία αρχιτεκτονική που λέγεται MaskGIT. Το MaskGIT[3] είναι μία σχετικά καινούργια προσέγγιση, τύπου Μετασχηματιστή, που προτάθηκε στα πλαίσια της σύνθεσης Εικόνας από Κείμενο. Είμαστε οι πρώτοι που χρησιμοποιούμε μία τέτοια μέθοδο για την ΟΙ. Συγκεκριμένα, σχηματίζουμε το βασικό μας μοντέλο προσθέτοντας στο πρωτότυπο MaskGIT επιπλέον υπο-στρώματα Ετέρο-Προσοχής (Cross-Attention), που του επιτρέπουν να ενσωματώνει πληροφορία από προηγούμενες και μελλοντικές περιγραφές εικόνων, όταν παράγει μία δεδομένη εικόνα.

Χτίζουμε πάνω σε αυτή την προσέγγιση με διάφορους τρόπους, σε αναζήτηση κατευθύνσεων που βελτιώνουν την επίδοση. Κάποια από τα πειράματά μας, όπως η χρήση προ-εκπαιδευμένου κωδικοποιητή κειμένου, η προσπάθεια απόπλεξης χαρακτηριστικών στον κρυφό χώρο, η χρήση Κριτή Συμβόλων (Token-Critic) και η αύξηση της ευκρίνεια του κρυφού χώρου χαρακτηριστικών, δεν έδωσαν καλύτερα αποτελέσματα.

Από την άλλη, εντοπίσαμε τρεις κατευθύνσεις που αποδείχτηκαν ωφέλιμες. Βρήκαμε ότι η προσθήκη SV-Στρωμάτων στον Μετασχηματισή βελτιώνει την επίδοση σε όλες τις μετρικές. Επιπλέον, προτείνουμε μία επιτυχημένη μέθοδο επαύξησης των γλωσσικών περιγραφών των εικόνων (captions), μέσω Μεγάλου Γλωσσικού Μοντέλου (LLM), η οποία είναι τυφλή ως προς τις εικόνες. Τέλος, η μέθοδος Καθοδήγησης Χαρακτήρων που προτείνουμε, βασιμένη τόσο στις θετικές όσο και στις αρνητικές υποδείξεις, επηρεάζει άμεσα την παραγωγή την βασικών Χαρακτήρων στις εικόνες και οδηγεί σε μεγάλη βελτίωση, ως προς όλες τις μετρικές. Συνδυάζουμε τις υποσχόμενες μεθόδους για να φτάσουμε στην καλύτερη αρχιτεκτονική μας: *MaskGST-CG$_{\pm}$ w/ aug. captions.*

Αξιολογούμε την προσέγγισή μας με βάση το Pororo-SV[18], που είναι το πίο διαδεδομένο σύνολο δεδομένων για αυτή την εργασία. Χρησιμοποιούμε τις πιο κυρίαρχες μετρικές στην προηγούμενη βιβλιογραφία, συμπεριλαμβανομένου των FID, Char-Acc, Char-F1 και BLEU-2/3. Το καλύτερο μοντέλο μας πετυχαίνει βέλτιστα αποτελέσματα (SOTA) στο Char-F1, Char-Acc και BLEU-2/3, τα οποία αναδεικνύουν ιδιαίτερα την αξία της μεθόδου Καθοδήγησης Χαρακτήρων. Επιπλέον, το μοντέλο μας ξεπερνά όλες τις προηγούμενες αρχιτεκτονικές GAN και Μετασχηματιστή, όσων αφορά το FID.

Λέξεις-κλειδιά — Οπτικοποίηση Ιστορίας, Τεχνητή Νοημοσύνη, Όραση Υπολογιστών, Επεξεργασία Φυσική Γλώσσας, Μετασχηματιστές, Επαύξηση γλωσσικών δεδομένων, Καθοδήγηση Χαρακτήρων

Abstract

Story Visualization (SV) is a challenging Artificial Intelligence task that falls in the intersection of Natural Language Processing (NLP) and Computer Vision. The task consists of generating a sequence of images that serve as a visualization of a given sequence of sentences. The sentences form a coherent narrative and so should the images. It was introduced in 2019[18] and has since been approached in multiple manners, including GANs [18, 36, 20, 19], Transformers[5, 1] and Diffusers[25, 33].

In this thesis we attempt to tackle the task based on an architecture called MaskGIT. MaskGIT[3] is a relatively recent Transformer-based approach, proposed for Text-to-Image synthesis. We are the first to employ this method for SV. Specifically, we form our baseline model by enhancing the original MaskGIT architecture with additional Cross-Attention sub-layers, that allow the model to integrate information from past and future captions, while generating an image.

We build on top of our baseline model in several different ways, in search of directions that improve performance. Some of our experiments, like leveraging a pre-trained text-encoder, attempting disentanglement in the latent space, using a Token-Critic and performing super-resolution in the latent space, do not yield better results.

On the other hand, we manage to detect three directions that prove beneficial. We find that adding SV-Layers to the Transformer improves its performance in all metrics. Additionally, we propose a successful, image-agnostic caption augmentation technique, that uses an LLM. Finally, our Character Guidance method, based on both positive and negative prompting, directly affects the generation of main Characters in the images and results in major improvements across all metrics. We combine promising approaches to arrive at our top-performing architecture; *MaskGST-CG$_{\pm}$ w/ aug. captions*.

We test our approach on Pororo-SV[18], which is the most widely adopted dataset for the task. We evaluate our models using the most prominent metrics in previous literature (including FID, Char-F1, Char-Acc and BLEU-2/3). Our best model achieves SOTA results in terms of Char-F1, Char-Acc and BLEU-2/3, which speaks of the merit of our Character Guidance approach. Additionally, it outperforms all previous GAN and Transformer approaches in terms of FID.

Keywords — Story Visualization, Artificial Intelligence, Computer Vision, NLP, Transformers, Caption Augmentation, Character Guidance

Ευχαριστίες

Πρώτον, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, τον κύριο Στάμου που μου εμπιστεύτηκε την εκπόνηση αυτής της διπλωματικής εργασίας. Ιδιαίτερα, θέλω να ευχαριστήσω την Μαρία Λυμπεραίου και τον Γιώργο Φιλανδριανό για την υπομονή τους, την συνεργασία και την καθοδήγησή τους σε όλη αυτή την διαδικασία. Επίσης, ευχαριστώ την κυρία Τζούβελη, που μου διέθεσε, μέσω του λογαριασμού της πρόσβαση στους υπολογιστικούς πόρους του συστήματος ARIS (GRNET).

Στα πιο προσωπικά, θέλω να ευχαριστήσω την οικογενειά μου και τους φίλους μου που ήταν δίπλα μου όλα τα χρόνια των σπουδών μου, κάποιοι εξ αυτών και πολύ πριν από αυτό. Η συμπαραστασή τους ήταν απαραίτητη για να φτάσω ως εδώ. Τέλος, θέλω να ευχαριστήσω τον Θεό που επιτρέπει να κλείσει αισίως και αυτό το κεφάλαιο.

Χρήστος Παπαδημητρίου, Μάρτιος 2024

Contents

List of Figures

List of Tables

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Εισαγωγή

Μέσα στην τελευταία δεκαετία, η εξέλιξη στο υλικό και το λογισμικό των υπολογιστών, η συσσώρευση μεγάλων ποσοτήτων δεδομένων, αλλά και η έρευνα στον τομέα της βαθιάς μάθησης έχουν οδηγήσει σε μια ανεπανάληπτη ανάπτυξη και υιοθέτηση συστημάτων τεχνητής νοημοσύνης (AI). Ειδικά, το 2023 έγινε φανερό ότι αυτό το φαινόμενο δικαίως χαρακτηρίζεται ως η επόμενη μεγάλη τεχνολογική επανάσταση. Η κυκλοφορία της, διάσημης πλέον, εφαρμογής ChatGPT στα τέλη του 2022 και η ταχύτατη άνοδός του σε δημοφιλία, πήραν την τεχνητή νοημοσύνη από τον καθαρά επιστημονικό κόσμο και την έφεραν στην δημόσια σκηνή. Η εφαρμογή αυτή έγινε μέρος την καθημερινότητάς, φαινόμενο της pop κουλτούρας και αντικείμενο πολιτικών συζητήσεων. Το ChatGPT, ανάμεσα σε άλλα LLMs (Μεγάλα Γλωσσικά Μοντέλα) (π.χ. [13, 37]) έχει επιδείξει ανεπανάληπτα αποτελέσματα, όσον αφορά την κατανόηση της φυσικής γλώσσας, αλλά και την παραγωγή της, μερικές φορές σε επίπεδο που συναγωνίζεται του ανθρώπους.

1.1.1 Παραγωγή Εικόνας από Κείμενο

Εκτός από την φυσική γλώσσα, οι εικόνες είναι μια από τις σημαντικότερες περιοχές της ανθρώπινης εμπειρίας, ενώ αποτελούν τρόπο επικοινωνίας και έκφρασης. Σε αυτό το πλαίσιο, η κατανόηση των εικόνων, αλλά και τη παραγωγή τους είναι εξίσου σημαντικές προκλήσεις για την Τεχνητή Νοημοσύνη, καθώς αυτή πλησιάζει στην έννοια που αποκαλούμαι *νοημοσύνη*, για τους ανθρώπους. Όσον αφορά την παραγωγή εικόνας από κείμενο, έχουνε γίνει αξιοσημείωτες προσπάθεις τα προηγούμενα χρόνια, με αφετηρία τα Γεννητικά Αντιθετικά Δίκτυα (GANs) που εμφανίστηκαν το 2014 [11] και κυριάρχησαν στην παραγωγή εικόνας για κάποια χρόνια [45, 46]. Πιο πρόσφατα, οι Μετασχηματιστές (Transformers) [26, 3, 4], αλλά και τα Μοντέλα Διάχυσης (Diffusion Models) [27, 30, 28] έχουν φέρει επανάσταση στην παραγωγή εικόνας, βελτιώνοντας την ποιότητα, αλλά και το εύρος των οπτικών θεμάτων που μπορούν να παραχθούν.

1.1.2 Οπτικοποίηση Ιστορίας

Σε αυτή την διπλωματική εργασία εστιάζουμε στην Οπτικοποίηση Ιστορίας (Story Visulization - SV). Ο σκοπός στην SV είναι η παραγωγή μιας ακολουθίας από εικόνες, κάθε μιά από τις οποίες αντιστοιχεί σε μια πρόταση, από μια ακολουθία δεδομένων προτάσεων. Οι προτάσεις, αυτές σχηματίζουν μια ενιαία αφήγηση. Το αντικείμενο αυτό προτάθηκε πρώτη φορά το 2019 από τους συγγραφείς του [18], οι οποίοι πρότειναν και το StoryGAN, το πρώτο μοντέλο για αυτή την εργασία.

Κατά μία έννοια, μπορούμε να δούμε την SV σαν επέκταση της Παραγωγής Εικόνας από Κείμενο, με προσθήκη μιας χρονικής διάστασης στο πρόβλημα. Εναλλακτικά μπορούμε να την δούμε ως ενδιάμεσο βήμα προς την Παραγωγή Βίντεο μεγάλου μήκους, από κείμενο, αφού η SV ασχολείται με μικρές ακολουθίες εικόνων (συνήθως 4-5), ενώ μια ταινία κανικού μήκους έχει χιλιάδες σκηνές.

Υπάρχουν δύο μεγάλες προκλήσεις στην SV. Πρώτον, στην παραγώμενη ακολουθία εικόνων, κάθε εικόνα πρέπει να περιέχει τα αντικείμενα που περιέχονται στην αντίστοιχη περιγραφή και να τα απεικονίζει ευκρινώς. Δεύτερον,

υπάρχει η αφηγηματική πλευρά: Αντικείμενα που εμφανίζονται σε πάνω από μία εικόνα πρέπει να παραμένουν συνεπή ως προς την εμφάνιση, ώστε να είναι αναγνωρίσιμα ως το ίδιο αντικείμενο. Τα πιο σημαντικά τέτοια αντικείμενα είναι οι βασικοί χαρακτήρες που εμφανίζονται στις ιστορίες.

1.1.3 Συνεισφορά

Η κύρια συνεισφορά μας είναι η υιοθέτηση ενός μοντέλου τύπου MaskGIT [3] για την Οπτικοποίηση Ιστορίας. Είμαστε οι πρώτοι που δομικάζουμε αυτή την αρχιτεκτονική για την συγκεκριμένη εργασία. Μάλιστα κρίνουμε ότι η συγκεκριμένη αρχιτεκτονική έχει λάβει δυσανάλογα μειωμένη προσοχή σε σχέση με τα μοντέλα Διάχυσης (Diffusion), ακόμα και στην συγγενή εργασία της παραγωγή εικόνας από κέιμενο.

Χρησιμοποιούμε μια ελαφριά παραλλαγή της αρχιτεκτονικής MaskGIT ως τον πυρήνα της δουλειάς μας. Πειραματιζόμαστε με ποικίλες τροποποιήσεις σε διάφορα κομμάτια της αρχιτεκτονικής, προσπαθώντας να καταλήξουμε σε ερευνητικές κατευθύνσεις που βελτιώνουν της ποιότητα των αποτελεσμάτων μας.

1.2 Προηγούμενες Τεχνικές στην Οπτικοποίηση Ιστορίας

Υπάρχουν αρκετές προηγούμενες δουλειές που εστιάζουν στην Οπτικοποίηση Ιστορίας.

Το StoryGAN[18] χρησιμοποιεί έναν Κωδικοποιητή Ιστορίας, ένας Αναδρομικό Κωδικοποιητή Κειμένου, για να διατηρεί συγκείμενο από προηγούμενα βήματα, καθώς και δύο ξεχωριστούς διευκρινιστές: Εικόνας και Ιστορίας. Διάφορα GANs μετά από αυτό χτίζουν πάνω σε αυτή την προσέγγιση με διαφορετικούς τρόπους. Το CP-CSV[36] προσθέτει έναν επιπλέον Παραγωγό που παράγει μάσκες διαχωρισμού προσκηνίου-παρασκηνίου, για να βοηθήσει με την παραγωγή των Χαρακτήρων. το DUCO-StoryGAN[20] προσθέτει Δυική Μάθηση (Dual Learning) μέσω επανα-περιγραφής βίντεο και έναν μηχανισμό Αντιγραφής-Μετασχηματισμού που χρησιμοποιεί εικόνες που παρήχθησαν στα προηγούμενα βήματα, για να ενημερώσει την εικόνα στο τρέχον βήμα. Το VLC-StoryGAN[19] ενσωματώνει οπτικό-χωρική πληροφορία, χρησιμοποιώντας πυκνό υποτιτλισμό (dense captioning) και χρησιμοποιεί γνώση κοινής λογικής, μέσω γράφων γνώσης.

Όσον αφορά τους Μετασχηματιστές, έχουν δημοσιευθεί δύο προσεγγίσεις. Το VP-CSV[5] προτείνει μία προσέγγιση δύο επιπέδων. Στο πρώτο επίπεδο, ένας Μετασχηατιστής προβλέπει οπτικά σύμβολα για περιοχές της εικόνας που αντιστοιχούν σε χαρακτήρες, ενώ στο δεύτερο επίπεδο, ένας άλλος Μετασχηματιστής, συμπληρώνει τα υπόλοιπα οπτικά σύμβολα που αντιστοιχούν στο παρασκήνιο. Το CMOTA [1] χρησιμοποιεί μονάδες μνήμης για να ενισχύσει την συνάφεια μεταξύ εικόνων της ίδιας ιστορίας. Επίσης προτείνει μία προσέγγιση δύο κατευθύνσεων (Κείμενο-προς-Εικόνα και Εικόνα-προς-Κείμενο) που του επιτρέπει να πραγματοποιεί επαύξηση των κειμενικών περιγραφών παράλληλα με την εκπαίδευση.

Αναφορικά με τα μοντέλα διάχυσης, υπάρχουν τρεις πρόσφατες τεχνικές που χρησιμοποιούν το προεκπαιδευμένο LDM[28]. Το AR-LDM[22] μοντελοποιεί την παραγωγή ακολουθίας εικόνων με αυτο-τροφοδοτικό (auto-regressive) τρόπο. Συγκεκριμένα, οι εικόνες παράγονται από ένα μοντέλο διάχυσης κρυφού χώρου (Latent Diffusion Model - LDM)[28], μία-μία, από την πρώτη, μέχρι την τελευταία. Η διαδικασία διάχυσης, χρησιμοποιεί ως συνθήκες την γλωσσική περιγραφή της τρέχουσας χρονικής στιγμής, καθώς και πολυτροπικές (multi-modal) αναπαραστάσεις από ζεύγη λεζάντας και παρηγμένης εικόνας, προηγούμενων χρονικών στιγμών. Η τρέχουσα λεζάντα κωδικοποιείται μέσω του CLIP[23], ενώ τα πολυτροπικά χαρακτηριστικά των προηγούμενων ζευγών εικόνας-λεζάντας σχηματίζονται μέσω του BLIP[17]. Το ACM-VSG[10] παρόμοια μοντελοποιεί των παραγωγή των εικόνων με αυτο-τροφοδοτικό τρόπο, χρησιμοποιώντας ως συνθήκη πολυτροπικά χαρακτηριστικά από προηγούμενα βήματα, μαζί με την τρέχουσα λεζάντα. Ωστόσο, εισάγει έναν επιπλέον μηχανισμό προσαρμοστικής καθοδήγησης (adaptive guidance) που αποσκοπεί στην εξώθηση εικόνων που έχουν παρόμοιες περιγραφές, να είναι κι αυτές οπτικά παρόμοιες. Το Causal-Story[35] βελτιώνει το AR-LDM εισάγοντας μία τοπική αιτιακή μάσκα προσοχής που περιορίζει το μέγεθος των αναπαραστάσεων που αφορούν τα προηγούμενα βήματα στην ιστορία, για να μετριάσει το πρόβλημα που δημιουργείται από συγκρουόμενες περιγραφές.

Τέλος, υπάρχουν δύο ακόμα τεχνικές που βασίζονται σε μοντέλα διάχυσης. Το StoryLDM[25] τροποποιεί και προσαρμόζει (fine-tune) το Stable Diffusion ώστε να χρησιμοποιηθεί με αυτο-τροφοδοτικό τρόπο, για την Οπτικοποίηση Ιστορίας. Πιο πρόσφατα, το StoryGPT-V[33] τροποποιεί το Stable Diffusion ώστε να εστιάζει στην παραγωγή των χαρακτήρων. Επίσης το ευθυγραμμίζει με ένα LLM, το οποίο βοηθάει στην αποσαφήνιση αναφορών Χαρακτήρων σε διαφορετικές περιγραφές της ίδιας ιστορίας. Σημειώνουμε ότι οι δύο τελευταίες

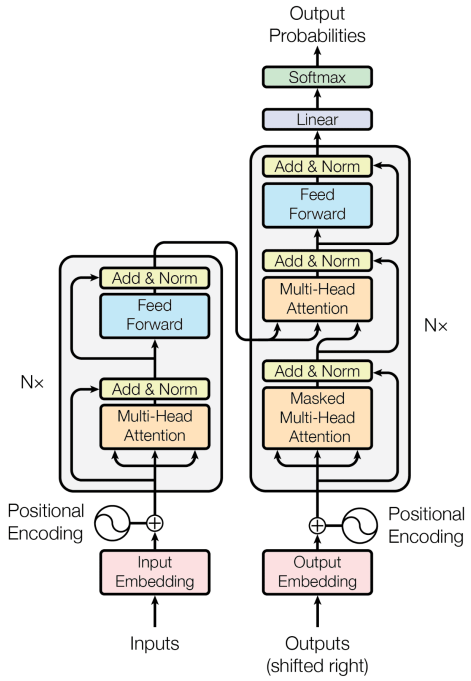


Figure 1.3.1: Η αρχιτεκτονική του Μετασχηματιστή[42]

μέθοδοι εφαρμόζονται σε μία παραλαγμένη, πιο δύσκολη έκδοση της SV, όπου επαναλαμβανόμενες αναφορές των ονομάτων των Χαρακτήρων αντικαθίστανται με αντωνυμίες (π.χ. αυτός, αυτή, αυτοί).

1.3 Θεωρητικό Υπόβαθρο

1.3.1 Μετασχηματιστές

Ένα από τα βασικά μοντέλα που συνδυάζει η προσέγγισή μας είναι ο Μετασχηματιστής. Οι Μετασχηματιστές (Transformers) προέρχονται ως μοντέλα από το πεδίο της Επεξεργασίας Φυσικής Γλώσσας. Η πρωτότυπη αρχιτεκτονική φαίνεται στην εικόνα 1.3.1. Τα δύο βασικά κομμάτια της αρχιτεκτονικής είναι ο Κωδικοποιητής (Encoder) και ο Αποκωδικοποιητής (Decoder).

Κωδικοποιητής

Ο Κωδικοποιητής (Encoder, Εικόνα 1.3.1 (αριστερά)) αποτελείται από 6 ακολουθιακά, όμοια στρώματα. Κάθε στρώμα αποτελείται από δύο υποστρώματα. Το πρώτο είναι ένα ένα υποστρώμα Αυτο-Προσοχής (Self-Attention) με πολλαπλές κεφαλές. Το δεύτερο είναι ένα Πλήρως Συνδεδεμένο Προς-τα-Εμπρός δίκτυο (fully connected feed-forward network). Υπάρχει Υπολειμματική Σύνδεση (Residual Connection) γύρω από κάθε υποστρώμα, ακολουθούμενη από Κανονικοποίηση Στρώματος (Layer Normalization). Η έξοδος κάθε υποστρώματος μπορεί να γραφτεί ως $LayerNorm(x + Sublayer(x))$, όπου $Sublayer(x)$ είναι η λειτουργία του συγκεκριμένου υποστρώματος.

Αποκωδικοποιητής

Ο Αποκωδικοποιητής (Decoder, Εικόνα 1.3.1 (δεξιά)) αποτελείται κι αυτός από 6 ακολουθιακά, όμοια στρώματα. Καθένα από αυτά έχει 3 υποστρώματα. Τα δύο είναι ίδια με του Κωδικοποιητή (Αυτο-Προσοχή και Πλήρως Συνδεδεμένο Προς-τα-Εμπρός Δίκτυο). Ανάμεσα σε αυτά τα δύο, εισάγεται ένα υποστρώμα Ετερο-Προσοχής Κωδικοποιητή-Αυτοκωδικοποιητή. Υιοθετούνται και πάλι Υπολειμματικές Συνδέσεις και Κανονικοποίηση Στρώματος. Επιπλέον, ο μηχανισμός Αυτο-Προσοχής εμποδίζεται από το να ασκήσει προσοχή σε επόμενες (μελλοντικές) θέσεις.

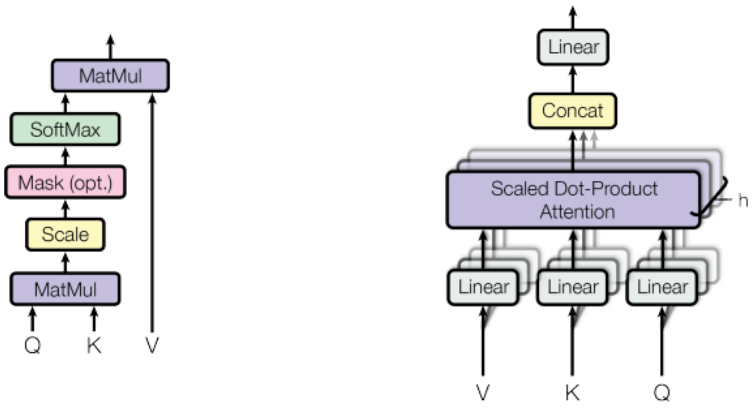


Figure 1.3.2: Κλιμακωμένη Προσοχή Εσωτερικού Γινομένου (αριστερά) και Προσοχή Πολλαπλών Κεφαλών (δεξιά) [42]

Μηχανισμοί Προσοχής

Κλιμακωμένη Προσοχή Εσωτερικού Γινομένου Η Κλιμακωμένη Προσοχή Εσωτερικού (Scaled Dot-Product Attention) είναι ο μηχανισμός προσοχής που χρησιμοποιείται στην πρωτότυπη δημοσίευση. Ο υπολογισμός γίνεται για ένα σετ από Ερωτήματα (Queries) πακεταρισμένα σε έναν πίνακα Q. Επίσης χρησιμοποιούνται ένας πίνακας Κλειδιών K και Τιμών V. Η διαδικασία απεικονίζεται στην εικόνα 1.3.2. Ο Πίνακας Q πολλαπλασιάζεται με τον Πίνακα K. Το αποτέλεσμα κλιμακώνεται και ενδεχομένως υπόκειται σε Μάσκα (π.χ. για να εμποδίσουμε μια θέση να ασκήσει Προσοχή σε μελλοντική θέση). Το αποτέλεσμα περνάει από Softmax ώστε να λάβουμε Βαμολογίες Συμβατότητας των Ερωτημάτων (Q) με τα Κλειδιά (K). Το τελικό αποτέλεσμα λαμβάνεται πολλαπλασιάζοντας τον τελευτάιο πίνακα συμβαβότητας με τον πίνακα Τιμών (V). Ουσιαστικά, το κάθε Διάνυσμα-Τιμή (πίνακας V) ζυγίζεται από την συμβατότητα κάθε ερωτήματος με το Κλειδί που αντιστοιχεί στο συγκεκριμένο Διάνυσμα-Τιμή.

Μαθηματικά ο υπολογισμός είναι ο εξής:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1.3.1}$$

Προσοχή Πολλαπλών Κεφαλών Η Προσοχή Πολλαπλών Κεφαλών (Multi-Head Attention, Εικόνα 1.3.2 (δεξιά)) επεκτείνει την ιδέα της προσοχής, απεικονίζοντας τους πίνακες Q, K, V με πολλαπλούς γραμμικούς μετασχηματισμούς, ώστε να δωθέι περισσότερη ελευθερία στο μοντέλο για να μάθει πολλαπλές διαφορετικές αναπαραστάσεις, σε διαφορετικούς υποχώρους. Οι υπολογισμοί τροποποιούνται ως εξής:

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O \tag{1.3.2}$$

Όπου:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{1.3.3}$$

Όπου οι μετασχηματισμοί είναι πίνακες: $W_i^Q \in \mathbf{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbf{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbf{R}^{d_{model} \times d_v}$ και $W_i^O \in \mathbf{R}^{hd_v \times d_{model}}$.

1.3.2 Αυτοκωδικοποιητής Παραλλαγών με Διανυσματικό Κβαντισμό

Το δέυτερο βασικό μοντέλο που χρησιμοποιεί η προσέγγισή μας είναι ο Αυτοκωδικοποιητής Παραλλαγών με Διανυσματικό Κβαντισμό (VQ-VAE: Vector Quantization Variational AutoEncoder).

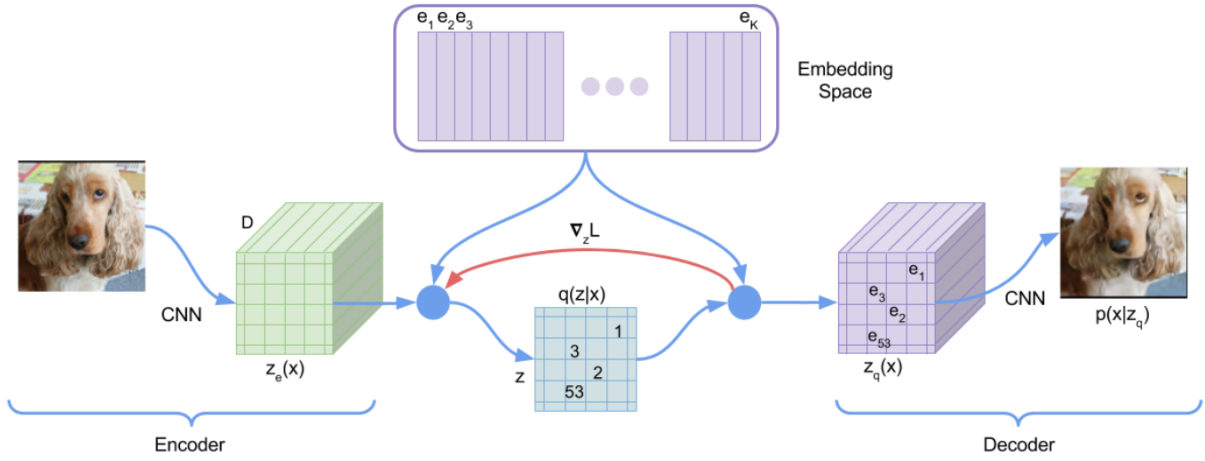


Figure 1.3.3: Η αρχιτεκτονική του VQ-VAE[39]

Κωδικοποιητής και Αποκωδικοποιητής

Ο VQ-VAE χρησιμοποιεί έναν Κωδικοποιητή ($p(z_e|x)$) και έναν Αποκωδικοποιητή ($p(\hat{x}|z_q)$) και οι δύο εκ των οποίων είναι Συνελικτικά Νευρωνικά Δίκτυα (CNNs). Ο Κωδικοποιητής απεικονίζει μία Εικόνα $x \in \mathbb{R}^{3\times N\times N}$ σε μια κρυφή (latent) αναπαράσταση $z_e \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}}$, όπου το f είναι παράγοντας συμπίεσης. Μετά τον κβαντισμο του z_e (βλ. επόμενη Ενότητα) ο Αποκωδικοποιητής απεικονίζει την κβαντισμένη κρυφή αναπαράσταση $z_q \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}}$ πίσω στον χώρο των εικόνων: $\mathbb{R}^{3\times N\times N}$.

Διακριτός Κρυφός Υποχώρος

Ο Διακριτός Κρυφός Υποχώρος (Discrete Latent Space) ορίζεται σαν ένας υποχώρος διανυσμάτων $e \in \mathbb{R}^{K\times D}$ όπου K είναι το πλήθος των κατηγοριών μιας κατηγορικής κατανομής και D η διαστατικότητα των διανυσμάτων. Όπως ειπώθηκε στην προηγούμενη υποενότητα, η έξοδος του Κωδικοποιητή, $z_e \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}}$ κβαντίζεται. Συγκεκριμένα, κάθε ένα από τα D-διάστατα διανύσματα ($(\frac{N}{f})^2$ το πλήθος) αντικαθίστανται από το κοντινότερό του στην βιβλιοθήκη διανυσμάτων e, με βάση αναζήτηση κοντινότερου γείτονα (Εξίσωση 1.3.4). Η κβαντισμένη εκδοχή της κρυφής αναπαράστασης δίνεται από την Εξίσωση 1.3.5. Σημειώνουμε ότι οι εξισώσεις αυτές αντιστοιχούν σε κρυφή αναπαράσταση ενός μοναδικού διανύσματος, ενώ για τις εικόνες χρησιμοποιείται ένας 2-D πίνακας από τέτοια διανύσματα. Από τις εξισώσεις προκύπτει ότι η κβαντισμένη αναπαράσταση, μπορεί να γραφτεί και σαν ένας πίνακας δεικτών, όπου κάθε δείκτης αντιστοιχεί σε ένα μοναδικό διάνυσμα στην βιβλιοθήκη διανυσμάτων e. Η ολοκληρωμένη λειτουργία του μοντέλου φαίνεται εποπτικά στην Εικόνα 1.3.3.

$$q(z=k,x) = \begin{cases} 1 & \text{for } k = argmin_j ||z_e(x) - e_j||_2 \\ 0 & \text{otherwise} \end{cases} \tag{1.3.4}$$

$$z_q(x) = e_k, \text{where } k = argmin_j ||z_e(x) - e_j||_2 \tag{1.3.5}$$

Πρότερη Κατανομή

Μετά την ολοκλήρωση της εκπαίδευσης του VQ-VAE, οι ερευνητές του πρωτότυπου μοντέλου εφαρμόζουν μία αυτοαναφορική (autoregressive) κατανομή, $p(z)$, πάνω στις διακριτές μεταβλητές z, ώστε να μπορεί να χρησιμοποιηθεί σαν παραγωγικό μοντέλο. Για αυτο τον σκοπό χρησιμοποιούν ένα PixelCNN [41, 40].

1.3.3 Μετασχηματιστές ως Πρότερες Κατανομές

Όπως ήδη αναφέραμε στην ενότητα 1.3.2, μπορούμε να χρησιμοποιήσουμε έναν VQ-VAE σαν παραγωγικό μοντέλο, αν του προσθέσουμε μια πρότερη κατανομή, πάνω στον κρυφό χώρο. Τα τελευταία χρόνια εμφανίστηκαν τεχνικές που χρησιμοποιούν Μετασχηματιστές σαν Πρότερες Κατανομές. Αρχικά χρησιμοποιήθηκαν

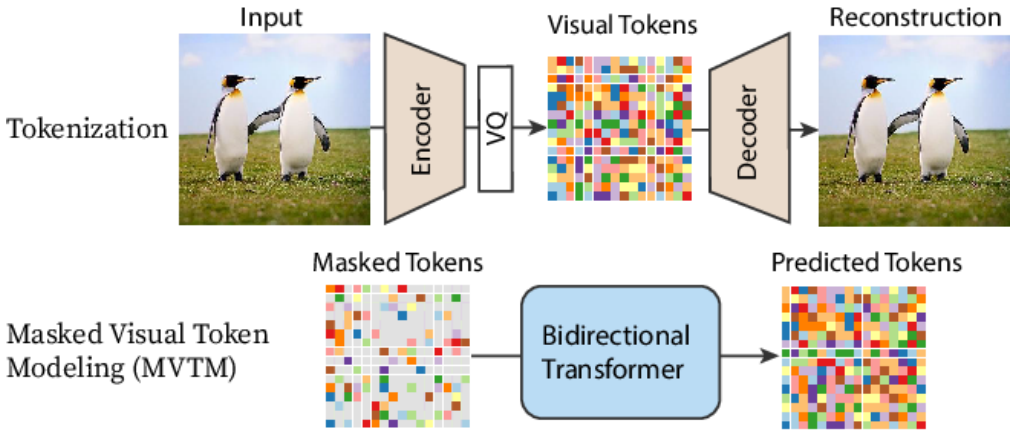


Figure 1.3.4: Ο τρόπος λειτουργίας του MaskGIT[3]

αυτοαναφορικοί (autoregressive) μετασχηματιστές (π.χ [26, 7, 8]). Πιο πρόσφατα εμφανίστηκαν πιο αποδοτικές τεχνικές που βασίζονται σε επαναληπτικούς, παράλληλους Μετασχηματιστές [3, 4].

MaskGIT

Το MaskGIT[3] (Masked Generative Image Transformer - Παραγωγικός Μετασχηματιστής Εικόνων με Μάσκες) εισήγαγε νέες τεχνικές τόσο στην εκπαίδευση, όσο και στον συμπερασμό ενός Μετασχηματιστή για παραγωγή εικόνων.

Πρώτο Επίπεδο Σε πρώτο επίπεδο, το MaskGIT υιοθετεί το VQ-GAN [8], έναν VQ-VAE με κάποιες παραλλαγές όσων αφορά την εκπαίδευση, για βελτιωμένα αποτελέσματα. Συγκεκριμένα, χρησιμοποιείται ένα VQ-GAN με βιβλιοθήκη 1024 διανυσμάτων. Ο παράγοντας συμπίεσης είναι $f = 16$, δηλαδή μία εικόνα ανάλυσης 256×256 απεικονίζεται σε ένα πλέγμα ανάλυσης $\frac{256}{16} \times \frac{256}{16} = 16 \times 16$, από διανύσματα.

Δεύτερο Επίπεδο Στο δεύτερο επίπεδο, εκπαιδεύεται ένας Μετασχηματιστής, ως Πρότερη Κατανομή πάνω στα οπτικά σύμβολα του κρυφού χώρου, υπό συνθήκη κειμένου. Η νέα τεχνική που χρησιμοποιείται στον Transformer λέγεται MVTM (Masked Visual Token Modelling - Μοντελοποίηση Μάσκας Οπτικού Συμβόλου).

MVTM κατά την εκπαίδευση Ας ονομάσουμε $Y = [y_i]_{i=1}^N$ τα σύμβολα (διανύσματα) του κρυφού χώρου που παράγονται από τον Κωδικοποιητή και τον Κβαντιστή του VQ-GAN, με μια εικόνα ως είσοδο. Έστω, επίσης $M = [m_i]_{i=1}^N$ μια δυαδική μάσκα $(0/1)$ για όλα τα σύμβολα. Σε κάθε βήμα εκπαίδευσης, δεισματολειπτούμε ένα υποσύνολο των συμβόλων και τα αντικαθιστούμε με ένα ειδικό σύμβολο $[MASK]$. $m_i = 1$ αντιστοιχεί σε σύμβολο που έχει αντικατασταθεί με μάσκα, ενώ για $m_i = 0$ το σύμβολο παραμένει ίδιο. Η προγραμματισμός των μασκών γίνεται μέσω μίας συνάρτησης $\gamma(r) \in (0, 1]$. Η διαδικασία λειτουργεί ως εξής:

- Ένα ποσοστό μεταξύ 0 και 1 δειγματοληπτείται μέση της $\gamma(r)$
- $\lceil \gamma(r) \cdot N \rceil$ σύμβολα επιλέγονται ομοιόμορφα και τους εφαρμόζεται η μάσκα
- Έστω $Y_{\bar{M}}$ ο πίνακας συμβόλων που προκύπτει μετά την εφαρμογή της μάσκας M πάνω στο Y
- Το μοντέλο εκπαιδεύεται για να ελαχιστοποιήσει την Αρνητική Λογαριθμική Πιθανοφάνεια (Negative Log-Likelihood):

$$\mathcal{L}_{mask} = -\mathbb{E}[\sum_{\forall i \in [1,N], m_i = 1} \log p(y_i | Y_{\bar{M}})] \tag{1.3.6}$$

Στην πραγματικότητα, οι πιθανότητες $p(y_i|Y_{\bar{M}}) \in \mathbb{R}^{N \times K}$ προβλέπονται από τον Transformer. Το K αντιστοιχεί στο μέγεθος της βιβλιοθήκης διανυσμάτων του VQ-GAN. Αυτά τα K διανύσματα είναι οι επιλογές από τις οποίες

ο Transformer καλείται να διαλέξει για κάθε οπτικό σύμβολο. Στην συνέχεια η ετερο-εντροπία (cross-entropy) μεταξύ των προβλεπόμενων κατανομών και των αληθινών one-hot διανυσμάτων υπολογίζεται για την εκπαίδευση του μοντέλου.

Επαναληπτικός Συμπερασμός Κατά τον συμπερασμό χρησιμοποιείται μια καινούργια μέθοδος, που περιλαμβάνει έναν μικρό αριθμό βημάτων, σε αντίθεση με τον παραδοσιακό, αυτοαναφορικό (autoregressive) συμπερασμό των Μετασχηματιστών που απαιτούσε τόσα βήματα όσο είναι το πλήθος των οπτικών συμβόλων στην αναπαράσταση την εικόνας, στον κρυφό χώρο. Μια οπτική σύγκριση των δύο τεχνικών φαίνεται στην εικόνα 1.3.5.

Όπως προκύπτει και από την περιγραφή της διαδικασίας εκπαίδευσης, δεν περιορίζουμε την Προσοχή (Attention) των οπτικών συμβόλων μόνο σε παρελθοντικά σύμβολα στην ακολουθία, εντός του Μετασχηματιστή (Μετασχηματιστής Δύο Κατευθύνσεων (Bidirectional)). Χάρη σε αυτό, θεωρητικά θα μπορούσαμε να προβλέψουμε όλα τα οπτικά σύμβολα για μια εικόνα με ένα, μοναδικό πέρασμα από τον Μετασχηματιστή. Στην πράξη, όμως αυτό δεν έχει καλά αποτελέσματα. Αντί αυτού, οι δημιουργοί του MaskGIT προτείνουν μια τεχνική που ξεκινά από έναν "άγραφο πίνακα" (όλα τα οπτικά σύμβολα έχουν αντικατασταθεί με μάσκα στο $Y_{\bar{M}}^{(0)}$). Ο αλγόριθμος στην επανάληψη t τρέχει ως εξής:

- Δεδομένων των Οπτικών Συμβόλων με εφαρμοσμένη μάσκα, στην συγκεκριμένη επανάληψη, $Y_{\bar{M}}^{(t)}$, οι πιθανότητες $p^{(t)} \in \mathbb{R}^{N \times K}$ για όλα τα σύμβολα που είχαν αντικατασταθεί με μάσκα προβλέπονται μέσω του Μετασχηματιστή.
- Σε κάθε θέση i όπου έχει εφαρμοστεί μάσκα, ένα σύμβολο $y_i^{(t)} = j$ δειγματοληπτείται με βάση τις πιθανότητες $p_i^{(t)} \in \mathbb{R}^K$ (αντιμετωπίζοντας το $p_i^{(t)}$ σαν πολυωνυμική (multinomial) κατανομή). Μετά την δειγματοληψία, η πιθανότητα $p_{ij}^{(t)}$ του συμβόλου (j) που επιλέχθηκε, χρησιμοποιείται ως βαθμός εμπιστοσύνης που δείχνει πόσο βέβαιο είναι το μοντέλο για αυτήν την πρόβλεψη. Για τις θέσεις όπου δεν είχε εφαρμοστεί μάσκα, ο βαθμός εμπιστοσύνης τίθεται στο 1.
- Ο αριθμός των συμβόλων στα οποία θα ξαναεφαρμοστεί μάσκα υπολογίζεται ως: $n = \lceil \gamma(\frac{1}{T})N \rceil$, όπου γ είναι η συνάρτηση προγραμματισμού μάσκας, N είναι ο συνολικός αριθμός από σύμβολα και T το συνολικό πλήθος των επαναλήψεων.
- Τα σύμβολα, με την νέα μάσκα εφαρμοσμένη, για την επόμενη επανάληψη, $Y_{\bar{M}}^{(t+1)}$ υπολογίζονται εφαρμόζοντας την νέα μάσκα $M^{(t+1)}$, που δίνεται από τον εξής τύπο:

$$m_i^{(t+1)} = \begin{cases} 1 & \text{if } c_i < sorted_j(c_j)[n] \\ 0 & \text{otherwise} \end{cases} \tag{1.3.7}$$

  όπου το c_i είναι ο βαθμός εμπιστοσύνης για το i-οστό σύμβολο.

Πιο συνοπτικά, το μοντέλο παράγει μια εικόνα σε T επεναλήψεις. Σε κάθε επανάληψη, προβλέπει όλα τα οπτικά σύμβολα και κρατάει αυτά για τα οποία έχει υψηλότερο βαθμό βεβαιότητας και ξαναεφαρμόζει μάσκα στα υπόλοιπα, ώστε να τα προβλέψει σε κάποια επόμενη επανάληψη. Το ποσοστό των συμβόλων στα οποία εφαρμόζεται μάσκα φθίνει σε κάθε επανάληψη, μέχρι που όλα τα σύμβολα προβλέπονται σε T επαναλήψεις.

Προγραμματισμός των Μασκών Όπως αναφέραμε, ο προγραμματισμός των μασκών γίνεται μέσω μίας συνάρτησης $\gamma(.)$. Κατά των εκπαίδευση, αυτή παίρνει σαν όρισμα ένα τυχαίο ποσοστό $r \in (0, 1]$, ενώ κατά τον συμπερασμό παίρνει τα: $0/T, 1/T, ..., (T-1)/T$, ανάλογα με την επανάληψη στην οποία βρισκόμαστε.

Η γ πρέπει να έχει τις εξής ιδιότητες:

- πρέπει να είναι συνεχής με τιμές $\in [0, 1]$, για ορίσματα $r \in [0, 1]$.
- Πρέπει αν είναι γνησίως φθίνουσα ως προς το r, με την ιδιότητα: $\gamma(0) \to 1$ and $\gamma(1) \to 0$. Αυτό είναι απαραίτητο για να συγκλίνει ο αλγόριθμος συμπερασμού, προβλέποντας όλα τα σύμβολα.

Οι ερευνητές του [3] πειραματίζονται με τρεις οικογένιες συναρτήσεων: *Γραμμικές*, *Κοίλες* (π.χ. Συνημίτονο) and *Κυρτές* (π.χ. Τετραγωνική). Παραδείγματα φαίνονται στην εικόνα 1.3.6. Οι ερευνητές αναφέρουν ότι καλύτερα αποτελέσματα δίνει το Συνημίτονο.

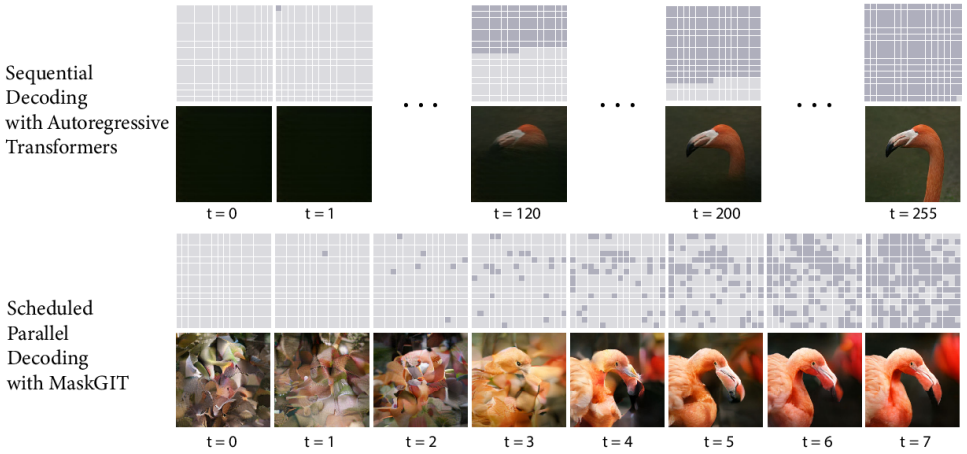


Figure 1.3.5: Αυτοαναφορικός Συμπερασμός vs Παράλληλος Επαναλήπτικός Συμπερασμός (MaskGIT) [3]

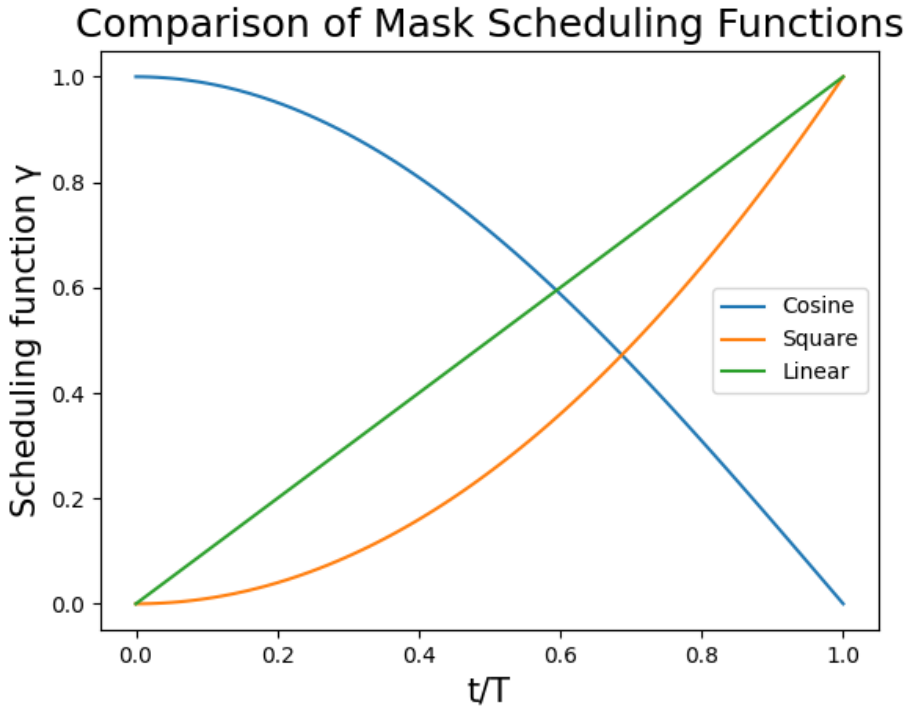


Figure 1.3.6: Σύγκριση Συναρτήσεων Προγραμματισμού Μάσκας

Αριθμός Επαναλήψεων Σχετικά με τον βέλτιστο αριθμό επαναλήψεων στον συμπερασμό, οι ερευνητές αναφέρουν ότι πέφτει ανέμεσα στις 8 και τις 12.

Κριτής Συμβόλων Ο Κριτής Συμβόλων [16] (Token-Critic) έχει προταθεί σαν βελτιωμένη τεχνική για δειγματοληψία των συμβόλων κατά των επαναληπτικό συμπερασμό του MaskGIT. Λειτουργεί σαν βοηθητικό μοντέλο που είναι υπεύθυνο για την εκτίμηση του βαθμού εμπιστοσύνης στα σύμβολα που προβλέπει το MaskGIT, σαν εναλλακτική στο να χρησιμοποιούνται οι πιθανότητες του MaskGIT σαν βαθμοί εμπιστοσύνης.

Ο Κριτής Συμβόλων είναι ένας Μετασχηματιστής που εκπαιδεύετε αφού ολοκληρωθεί η εκπαίδευση του MaskGIT. Κατά την **εκπαίδευση** $Y = [y_i]_{i=1}^N$ είναι τα σύμβολα του κρυφού χώρου κωδικοποιημένα από το VQ-GAN. $M = [m_i]_{i=1}^N$ είναι οι δυαδικές μάσκες για όλα τα σύμβολα. $Y_{\bar{M}}$ είναι το διάνυσμα συμβόλων, αφού εφαρμοστεί η μάσκα στο Y. Δεδομέων του $Y_{\bar{M}}$, δειγματολειπτούμε το $\hat{Y}$ από την $p(y_i|Y_{\bar{M}})$ (αυτή είναι η κατανομή που παραμετροποιεί το MaskGTI). Στην συνέχεια, σχηματίζουμε το $\tilde{Y} = \hat{Y} \odot (1 - M) + Y \odot M$. Το $\tilde{Y}$ είναι ισοδύναμο με το $Y_{\bar{M}}$, όπου όλα τα $[MASK]$ σύμβολα έχουν αντικατασταθεί από το σύμβολο που πρόβλεψε το MaskGIT σε εκείνη την θέση Ο Κριτής Συμβόλων εκπαιδεύεται για να ελαχιστοποιήσει το:

$$\mathcal{L} = \mathbb{E}[\sum_{j=1}^{N} BCE(m_j, p_\phi(m_j|\tilde{Y}, c))] \tag{1.3.8}$$

Όπου η $p_\phi(.)$ είναι η κατανομή που παραμετροποιεί ο Κριτής Συμβόλων. Δηλαδή ο Κριτής Συμβόλων μαθαίνει να προβλέπει την δυαδική μάσκα, με συνθήκη το $\tilde{Y}$ και το c, οπού το c μπορεί να είναι οποιαδήποτε συνθήκη (π.χ. συνθήκη κειμένου).

Κατά τον συμπερασμό χρησιμοποιείται η επαναληπτική λογική του MaskGIT, με μια μικρή τροποποίση. Ξεκινάμε με έναν "κενό πίνακα" $Y_{\bar{M}_1}^{(0)}$ (όλα τα σύμβολα είναι αντικατεστημένα από την μάσκα). Στην επανάληψη t δειγματοληπτούμε το $Y^{(t)}$ μέσω του MaskGIT, δηλαδή:

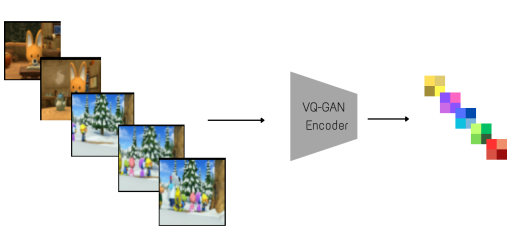
$$Y^{(t)} \sim p_\theta(Y^{(t)}|Y_{\bar{M}_t}^{(t-1)}, c) \tag{1.3.9}$$

όπου η κατανομή $p_\theta(.)$ είναι αυτή που έχει μάθει το MaskGIT.

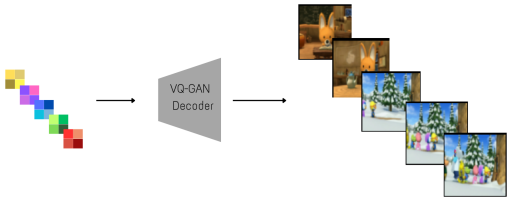
Τώρα, αντί να χρησιμοποιήσουμε τις πιθανότητες που έδωσε το MaskGIT σαν βαθμούς εμπιστοσύνης, με βάση τους οποίους θα ξαναεφαρμόσουμε μάσκες για την επόμενη επανάληψη, χρησιμοποιούμε τον Κριτή Συμβόλων. Δειγματοληπτούμε το $M_{t+1} \sim p_\phi(M_{t+1}|Y^{(t)}, c)$ ($p_\phi(.)$ είναι η κατανομή που έχει μάθει ο Κριτής Συμβόλων). Το $M_{t+1} \in \mathbb{R}^N$ περιέχει N τιμές, όλες μεταξύ 0 και 1 που αντιστοιχούν στους βαθμούς εμπιστοσύνης για κάθε σύμβολο. Δεδομένου ότι ο Κριτής Συμβόλων είναι Μετασχηματιστής, χρησιμοποιεί Προσοχή (Attention) για να λάβει υπόψιν την συσχέτιση μεταξή των συμβόλων όταν προβλέπει τους βαθμούς εμπιστοσύνης, κάτι που αναμένεται να βελτιώνει την ποιότητα της απόφασης, σε σχέση με την ανεξάρτητη δειγματοληψία του MaskGIT.

1.3.4 Επαύξηση Κειμενικών Δεδομένων με Χρήση LLM

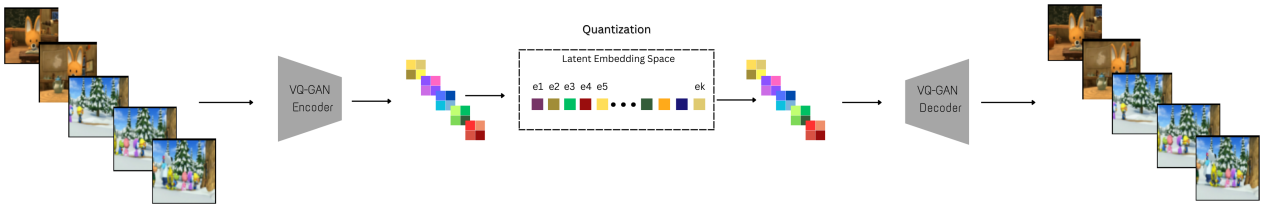
Οι εξαιρετικές δυνατότητες των Μεγάλων Γλωσσικών Μοντέλων (Large Language Models - LLMs) έχουν αξιοποιηθεί στο παρελθόν, στο πλαίσιο διαφόρων εργασιών[38, 43, 6, 44, 9]. Στο [9] προτείνεται μία μέθοδος επαύξησης περιγραφών εικόνων, σε ζεύγη λεζάντας-εικόνας τα οποία χρησιμοποιούνται για την εκπαίδευση ενός μοντέλου CLIP[23]. Αρχικά, παράγονται εναλλακτικές λεζάντες για έναν μικρό αριθμό ζευγαριών λεζάντας - εικόνας, μέσω διαφόρων μεθόδων, συμπεριλαμβανομένου ανθρώπινης εργασίας και chatbots. Η πρωτότυπες και οι εναλλακτικές περιγραφές συνδυάζονται για να δημιουργηθούν μετα-ζευγάρια εισόδου-εξόδου (meta-input-output pairs). Στην συνέχεια, χρησιμοποιείται το LLaMA[37] για να παραχθούν εναλλακτικές λεζάντες για όλα τα δείγματα του συνόλου δεδομένων. Τα μετα-ζευγάρια εισόδου-εξόδου χρησιμοποιούνται σε αυτή την διαδικασία σαν συγκείμενο (context), ώστε το LLM να κατανοήσει καλύτερα την εργασία.



(a) Οι εικόνες μιας ιστορίας απεικονίζονται στον κρυφό χώρο μέσω του Κωδικοποιητή του VQ-GAN



(b) Η κβαντισμένη αναπαράσταση της εικόνας στον κρυφό χώρο μπορεί να μετατραπεί σε εικόνα μέσω του Αποκωδικοποιητή του VQ-GAN



(c) Μεταξύ του Κωδικοποιητή και του Αποκωδικοποιητή οι κρυφές αναπαραστάσεις κβαντίζονται μέσω μιας βιβλιοθήκης διανυσμάτων

Figure 1.4.1: Η λειτουργία του VQ-GAN

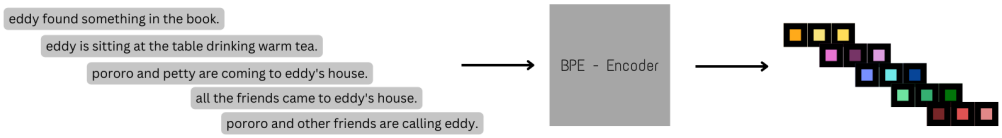


Figure 1.4.2: Οι προτάσεις μιας ιστορίας υπόκεινται κωδικοποίηση BPE[32] για να αντιστοιχιστούν σε διακριτά γλωσσικά σύμβολα (tokens)

1.4 Προτεινόμενες Τεχνικές

1.4.1 Κωδικοποίηση των Εικόνων

VQ-GAN

Για τον κβαντισμό των εικόνων χρησιμοποιούμε το VQ-GAN [8]. Δεδομένων των εικόνων τις ιστορίας: $X = \{X_1, X_2, ..., X_n\}$, μπορούμε να τις περάσουμε από τον Κωδικοποιητή του VQ-GAN (Εικόνα 1.4.1a) και να τις κβαντίσουμε για να παραχθούν τα αντίστοιχα διακριτά οπτικά σύμβολα $Z = \{Z_1, Z_2, ..., Z_n\}$. Στην συνέχεια ένας Μετασχηματιστής εκπαιδεύεται για να προβλέπει τα οπτικά σύμβολα, υπο συνθήκη κειμένου (βλ. Ενότητα 1.4.3). Αφού προβλεφθούν τα οπτικά σύμβολα, μπορούν να μεταφραστούν σε εικόνες, μέσω του Αποκωδικοποιητή του VQ-GAN (Εικόνα 1.4.1b).

1.4.2 Κωδικοποίηση των Γλωσσικών Περιγραφών

Ως προς την κωδικοποίηση γλωσσικών περιγραφών πειραματιζόμαστε με δύο μεθόδους: εξ αρχής εκπαίδευση διανυσμάτων λέξεων (word embeddings) ή χρήση διανυσμάτων λέξεων που έχουν εξαχθεί από κάποιο LLM. Ανεξάρτητα από την μέθοδο, κάθε μία γλωσσική περιγραφή μιας ιστορίας αντιστοιχίζεται σε μια ακολουθία από διανύσματα. Συμβολίζουμε τις κωδικοποιημένες περιγραφές μιας ιστορίας με $T = \{T_1, T_2, ..., T_n\}$.

Εξ Αρχής εκπαίδευση Διανυσμάτων Λέξεων

Όταν εκπαιδεύουμε διανύσματα λέξεων χρησιμοποιούμε κωδικοποίηση BPE, που απεικονίζει κάθε γλωσσική περιγραφή σε μια ακολουθία διακριτών συμβόλων (Figure 1.4.2). Κατά την κωδικοποίηση BPE χρησιμοποιούμε λεξιλόγια 2500 συμβόλων.

Χρήση LLM

Αντί να εκπαιδεύσουμε διανύσματα λέξεων από την αρχή, μπορούμε να χρησιμοποιήσουμε τα διανύσματα από ένα προ-εκπαιδευμένο LLM. Εμπνεόμενοι από το MUSE[4], πειραματιζόμαστε με το T5-XXL [24]. Για να παράγουμε διανύσματα από τις γλωσσικές περιγραφές, παιρνάμε κάθε περιγραφή από το T5 και αποσπούμε τα διανύσματα τις τελευταίας κρυφής κατάστασης του μοντέλου. Στην συνέχεια τα χρησιμοποιούμε σαν αναπαραστάσεις για τις γλωσσικές περιγραφές, στον Μετασχηματιστή μας.

1.4.3 MaskGST

Είσοδος

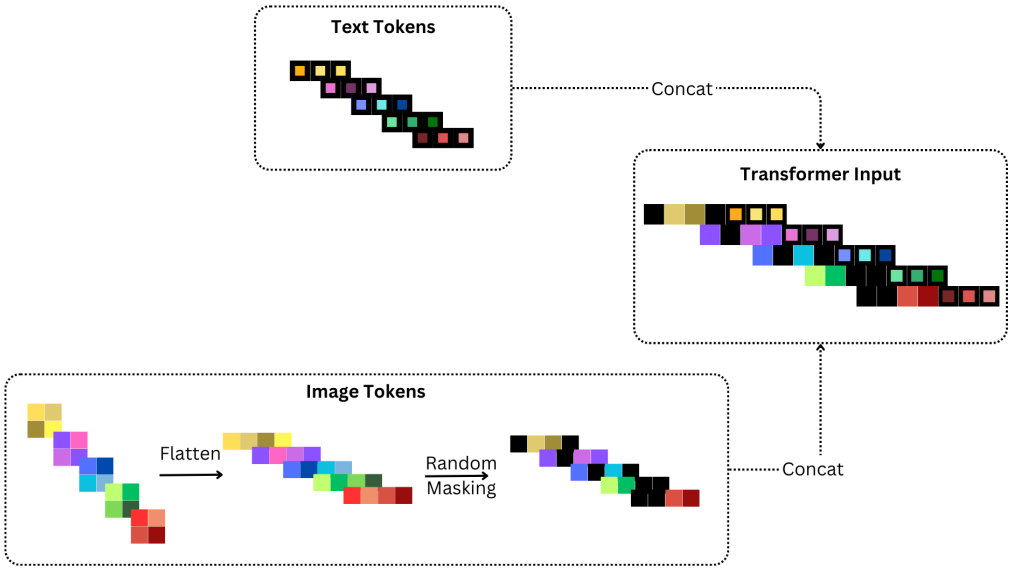


Figure 1.4.3: Η είσοδος του μετασχηματιστή είναι η συνένωση των οπτικών και των γλωσσικών συμβόλων (tokens) για κάθε ένα από τα ζευγάρια εικόνας/περιγραφής στην ιστορία (5). Στα οπτικά σύμβολα επιπεδώνονται (flatten) και τους εφαρμόζεται μάσκα (με τυχαίο τρόπο) πριν την συνένωση με τα γλωσσικά σύμβολα.

Δεδομένων των οπτικών συμβόλων $Z = \{Z_1, Z_2, ..., Z_n\}$ ($Z_i \in \mathbb{R}^{m \times m \times d}$) και των γλωσσικών συμβόλων $T = \{T_1, T_2, ..., T_n\}$ ($T_i \in \mathbb{R}^{l \times d}$), για μια ιστορία, σχηματίζουμε την είσοδο των Μετασχηματιστών μας όπως φαίνεται στην Εικόνα 1.4.3. Τα οπτικά σύμβολα $Z \in \mathbb{R}^{n \times m \times m \times d}$ ισοπεδώνονται (flatten) σε μια ακολουθία $Z' \in \mathbb{R}^{n \times (m \cdot m) \times d}$. Στην συνέχεια τους εφαρμόζουμε τυχαία μάσκες, όπως ακριβώς στο *MaskGIT* για να πάρουμε τα $\bar{Z} \in \mathbb{R}^{n \times (m \cdot m) \times d}$. Τέλος, κάθε αναπαράσταση εικόνας συνενώνεται με την αντίστοιχη γλωσσική περιγραφή για να σχηματιστούν τα $Input_i = (\bar{Z}_i; T_i)$. Η είσοδος των Μετασχηματιστών μπορεί να γραφτεί ως:

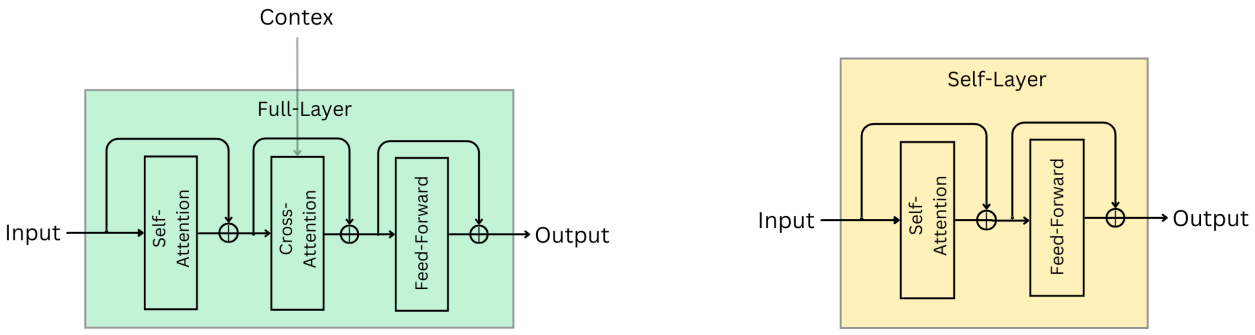
$$Input = \{Input_1, ..., Input_n\} \in \mathbb{R}^{n \times (m \cdot m + l) \times d} \tag{1.4.1}$$

όπου με l συμβολίζεται το μήκος της γλωσσικής αναπαράστασης, $m \times m$ είναι η ανάλυση των κρυφών αναπαραστάσεων των εικόνων, n είναι ο αριθμός των εικόνων μιας ιστορίας και d είναι η κρυφή διάσταση του Μετασχηματιστή.

Είδη Στρωμάτων Μετασχηματιστών

Παρακάτω περιγράφουμε τα είδη από Στρώματα Μετασχηματιστών που χρησιμοποιούμε στα μοντέλα μας.

Πλήρες Στρώμα Το *Πλήρες Στρώμα* 1.4.4a είναι ένα παραδοσιακό στρώμα ενός Αποκωδικοποιητή-Μετασχηματιστή, αποτελούμενο από τρία υποεπίπεδα: Αυτο-Προσοχή (Self-Attention), Ετερο-Προσοχή (Cross-Attention) και Πλήρως Συνδεδεμένο Προς-τα-Εμπρός Επίπεδο (Fully Connected Feed Forward Layer). Δε-



(a) Πλήρες Στρώμα (b) Αυτό-Στρώμα

Figure 1.4.4: Τα κύρια στρώματα που χρησιμοποιούνται στους Μετασχηματιστές μας

δομένης της εισόδου $I \in \mathbb{R}^{n \times a \times d}$ και κάποιο συγκείμενο (context) $C \in \mathbb{R}^{n \times c \times d}$, η έξοδος του στρώματας Αυτο-Προσοχής και Ετερο-Προσοχής, αντίστοιχα, μπορεί να υπολογιστεί ως εξής:

$$\begin{aligned} Self - Attention(I) &= MultiHead_1(Q = I, K = I, V = I) \\ Cross - Attention(I, C) &= MultiHead_2(Q = I, K = C, V = C) \end{aligned} \tag{1.4.2}$$

Το Feed-Forward υποστρώμα αποτελείται από ένα γραμμικό μετασχηματισμό, ακολουθούμενο από μια συνάρτηση ενεργοποίησης, ακολουθούμενη από Κανονικοποίηση Στρώματος (Layer Normalization). Όπως φαίνεται στην Εικόνα, υπάρχει υπολειμματική (residual) σύνδεση γύρω από κάθε υποεπίπεδο.

Αυτό-Στρώμα Όπως φαίνεται στην εικόνα 1.4.4b, το Αυτο-Επίπεδο είναι όμοιο με το Πλήρες Επίπεδο, με την διαφορά ότι παραλείπει το υποστρώμα Ετερο-Προσοχής και συνεπώς δεν χρησιμοποιεί συγκείμενο.

SV-Στρώμα Το SV-Στρώμα (Story Visualization Σρώμα) φαίνεται στην εικόνα 1.4.6. Αποτελείται από ένα Αυτο-Στρώμα, του οποίου προηγείται ένα υποστρώμα Προεπεξεργασίας και έπεται ένα στρώμα Μετα-επεξεργασίας.

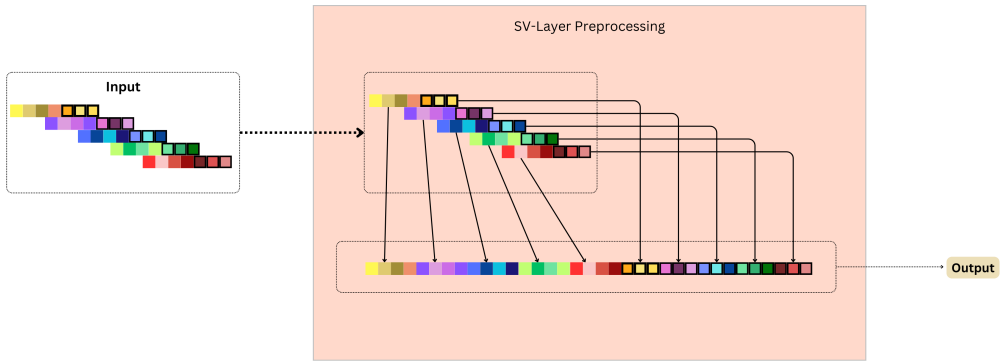
Το **Υποστρώμα Προεπεξεργασίας** παίρνει σαν είσοδο μια δέσμη δεδομένων (batch) $\in \mathbb{R}^{n \times (m \cdot m + l) \times d}$, που αντιστοιχεί σε στα n ζευγάρια εικόνας-περιγραφής σε μια ιστορία. Μετασχηματίζει την είσοδο σε μια αναπαράσταση $\in \mathbb{R}^{1 \times (n \cdot (m \cdot m) + n \cdot l) \times d}$ τοποθετόντας τα οπτικά σύμβολα για όλες τις εικόνες της ιστορίας, το ένα δίπλα στο άλλο και συνενώνοντας τα με την ακολουθία όλων των γλωσσικών συμβόλων της ιστορίας, τοποθετημένα το ένα δίπλα στο άλλο (Εικόνα 1.4.5a).

Το **Υποστρώμα Μετα-επεξεργασίας** έχει την ακριβώς αντίστροφη λειτουργία. Παίρνει μια είσοδο $\in \mathbb{R}^{1 \times (n \cdot (m \cdot m) + n \cdot l) \times d}$. Την μετασχηματίζει σε μια έξοδο $\in \mathbb{R}^{n \times (m \cdot m + l) \times d}$, όπου καθεμια από τις $((m \cdot m + l) \times d)$-μεγέθεις ακολουθίες στην δέσμη μεγέθους n αντιστοιχεί στα οπτικά σύμβολα μιας εικόνας, συνενωμένα με τα αντίστοιχα γλωσσικά σύμβολα (Εικόνα 1.4.5b).

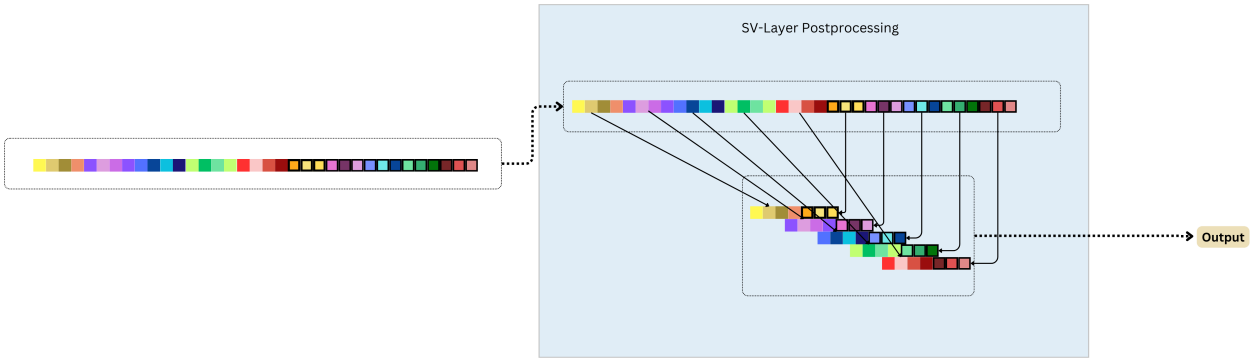
Σε διαισθητικό επίπεδο, το υποστρώμα Προεπεξεργασίας φέρνει την αναπαράσταση της ιστορίας από ένα φορμάτ εικόνα-εικόνα σε ένα φορμάτ σε επίπεδο ιστορίας, όπου όλη η ιστορία αντιμετωπίζεται σαν μια συνεχής ακολουθία συμβόλων, που αντιστοιχούν στις εικόνες και τις περιγραφές. Με αυτόν τον τρόπο, στο Αυτο-Στρώμα που ακολουθεί, τα σύμβολα, από οποιοδήποτε από τις n θέσεις στην ιστορία μπορούν να ασκήσουν Προσοχή (Attention) σε σύμβολα οποιασδήποτε άλλης θέσης και να ενσωματώσουν σχετική πληροφορία. Ακολούθως, φέρνουμε τα σύμβολα πίσω στο αρχικό εικόνα-εικόνα φορμάτ με το υποστρώμα Μετα-επεξεργασίας.

Προτεινόμενα μοντέλα τύπου Μετασχηματιστή

MaskGST Ονομάζουμε την προτεινόμενη αρχιτεκτονική μας MaskGST (Masked Generative Story Transformer - Παραγωγικός Μετασχηματιστής Ιστοριών με Μάσκες) για να τονίσουμε ότι βασίζεται στο MaskGIT.



(a) Προεπεξεργασία για το SV-Στρώμα



(b) Μετά-επεξεργασία για το SV-Στρώμα

Figure 1.4.5: Επεξεργασία δεδομένων για το SV-Στρώμα

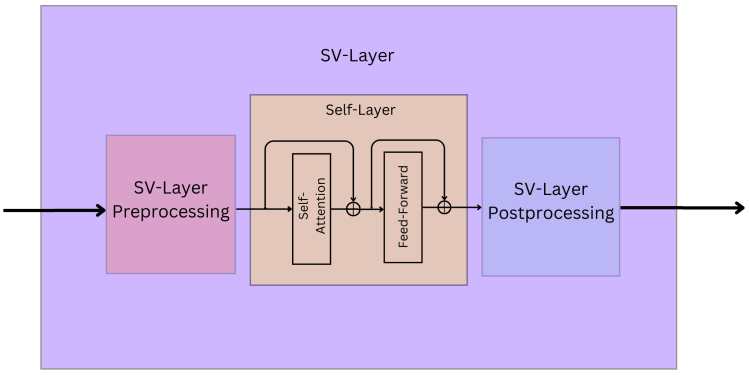


Figure 1.4.6: SV-Στρώμα

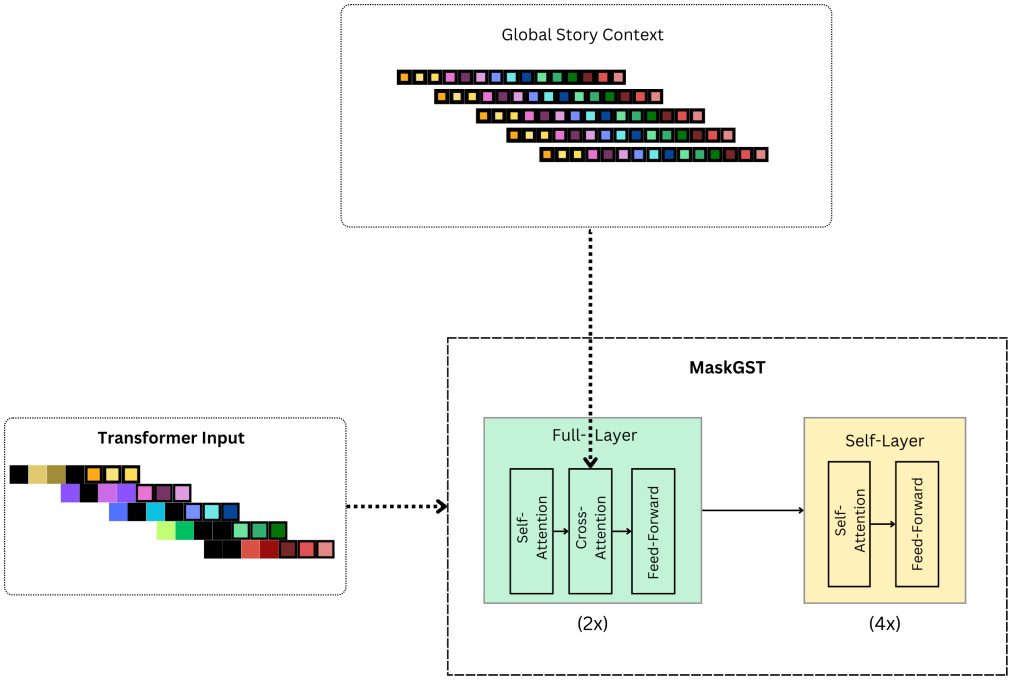


Figure 1.4.7: Το MaskGST

Το MaskGST αποτελείται από δύο Πλήρη Στρώματα, που ακολουθούνται από κάποια Αυτο-Στρώματα. Η Εικόνα 1.4.7 δείχνει μια έκδοση του μοντέλου με 4 Αυτο-Στρώματα (6 στρώματα συνολικά).

Το συγκείμενο που χρησιμοποιείται από τα υποστρώματα Ετερο-Προσοχής αποτελείται από όλες τις γλωσσικές περιγραφές στην ιστορία. Δηλαδή, για να προβλέψουμε τα οπτικά σύμβολα για κάθε εικόνα, πραγματοποιούμαι Ετερο-Προσοχή με όλες τις άλλες γλωσσικές περιγραφές της ιστορίας. Με αυτόν τον τρόπο, επιτρέπουμε στο μοντέλο να ϋιοθετήσει χρήσιμη, σχετική πληροφορία από προηγούμενα και επόμενα χρονικά σημεία στην ιστορία.

MaskGST-SV Το MaskGST-SV (MaskGST Story Visualization) χρησιμοποιεί δύο Πλήρη Στρώματα στην αρχή της παραγωγικής διαδικασίας. Αυτά ακολουθούνται προαιρετικά από μερικά Αυτο-Στρώματα. Στην συνέχεια, έχουμε δύο SV-Στρώματα, τα οποία μπορούν προαιρετικά να ακολουθούνται από κάποια ακόμα Αυτο-Στρώματα. Για παράδειγμα, στην Εικόνα 1.4.8 φαίνεται ένα MaskGST-SV που τοποθετή δύο Αυτο-Στρώματα ανάμεσα στα Πλήρη Στρώματα και τα SV-Στρώματα. Επίσης τοποθετεί δύο Αυτο-Στρώματα και μετά τα SV-Στρώματα (8 Στρώματα συνολικά).

Εκπαίδευση Όλα οι προτεινόμενοι μετασχηματιστές, μεταρέπουν την είσοδό τους: $Input = \{Input_1, ..., Input_n\} \in \mathbb{R}^{n \times (m \cdot m + l) \times d}$ σε μία έξοδο, ίδιων διαστάσεων:

$$Output = \{Output_1, ..., Output_n\} \in \mathbb{R}^{n \times (m \cdot m + l) \times d} \tag{1.4.3}$$

Για κάθε αντικείμενο στην n-δέσμη (n-batch), αφήνουμε τα l σύμβολα που αντιστοιχούν στην κειμενική περιγραφή και κρατάμε τα $m \cdot m$ οπτικά σύμβολα. Συνεπώς, έχουμε:

$$\overline{Output} = \{\overline{Output_1}, ..., \overline{Output_n}\} \in \mathbb{R}^{n \times (m \cdot m) \times d} \tag{1.4.4}$$

Η έξοδος αυτή περνάει από ένα γραμμικό μετασχηματισμό για να πάρουμε πιθανότητες:

$$y = \mathbf{Lin}(\overline{Output}) \in \mathbb{R}^{n \times (m \cdot m) \times K} \tag{1.4.5}$$

Όπου K είναι το πλήθος των διανυσμάτων που διαθέτει η διακριτή βιβλιοθήκη του VQ-GAN. Τέλος, οι παραπάνω πιθανότητες μπορούν να χρησιμοποιηθούν για να εκπαιδεύσουμε το μοντέλο, συκρίνοντάς τες με τα πραγματικά

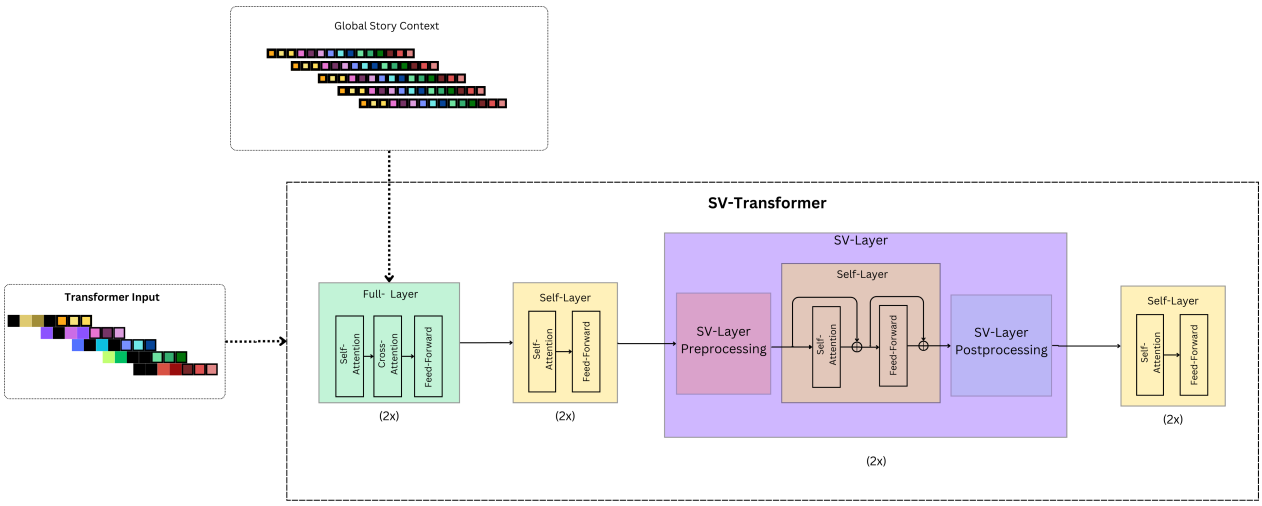


Figure 1.4.8: MaskGST-SV

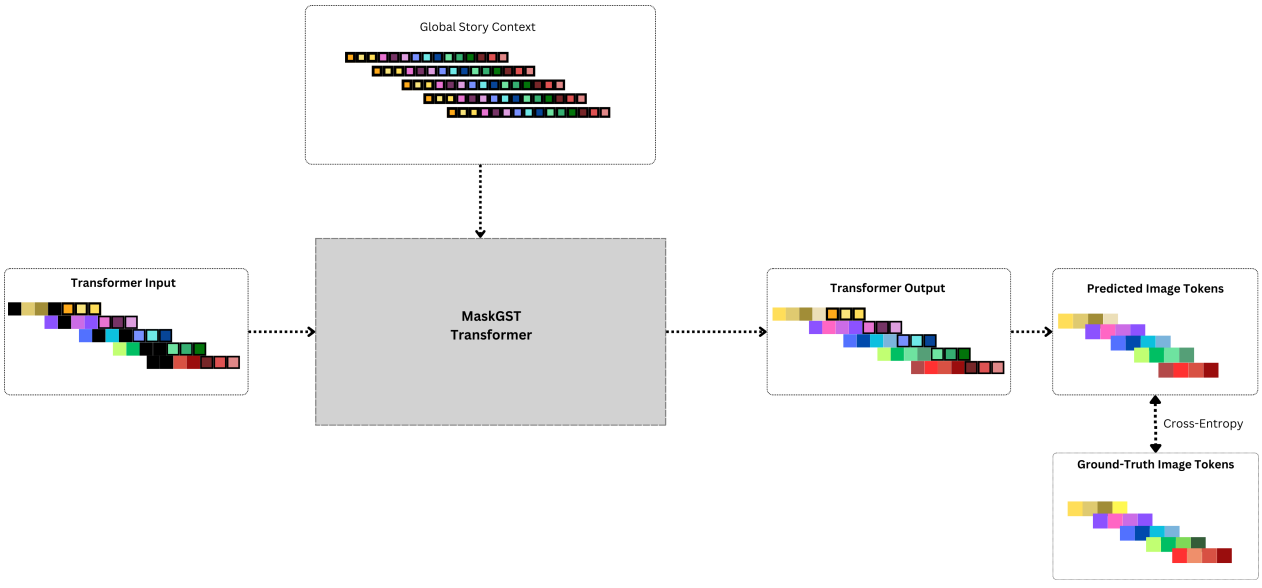


Figure 1.4.9: Η διαδικασία εκπαίδευσης του MaskGST

(ground-truth) διανύσματα που αντιστοιχούν στα πραγματικά οπτικά σύμβολα. Για την εκπαίδευση, χρησιμοποιούμε την τεχνική MVTM (Masked Visual Token Modeling) που προτείνεται στο MaskGIT (Ενότητα 1.3.3).

Συμπερασμός Όπως στην εκπαίδευση, έτσι και στον συμπερασμό ϋιοθετούμε τον παράλληλο, επαναληπτικό αλγόριθμο που προτείνεται στο MaskGIT (Ενότητα 1.3.3).

Καθοδήγηση Χαρακτήρων

Προτείνουμε μια νέα τεχνική για να βελτιώσουμε την παραγωγή των χαρακτήρων στις ιστορίες. Προσθέτουμε στο μοντέλο μια βιβλιοθήκη από $2 \cdot C_n$ επιλέον Διανύσματα (Embeddings) Χαρακτήρων, όπου C_n είναι ο αριθμός των βασικών χαρακτήρων που είναι παρόντες στο Dataset (Για το Pororo-SV έχουμε $C_n = 9$). Για κάθε χαρακτήρα έχουμε ένα Θετικό Διάνυσμα (ο χαρακτήρας αναφέρεται στην γλωσσική περιγραφή) και ένα Αρνητικό Διάνυσμα (ο χαρακτήρας δεν αναφέρεται). Όταν χρησιμοποιούμε αυτή την τεχνική, συνενώνουμε C_n διανύσματα στην είσοδο του Μετασχηματιστή, ένα για καθε χαρακτήρα. Θετικά Διανύσματα χρησιμοποιούνται για τους χαρακτήρες και που είναι παρόντες στην τρέχουσα γλωσσική περιγραφή και Αρνητικά για τους υπόλοιπους. Σε αυτή την περίπτωση, η είσοδος του Μετασχηματιστή γίνεται:

$$Input = \{Input_1, ..., Input_n\} \in \mathbb{R}^{n \times (m \cdot m + l + C_n) \times d} \tag{1.4.6}$$

όπου: $Input_i = (\bar{Z}_i; T_i; C_i)$ και $C_i = \{pos_{char}\}_{\text{char} \in T_i} \bigcup \{neg_{char}\}_{\text{char} \notin T_i}$. το pos_{char} αντιπροσωπεύει το Θετικό Διάνυσμα για τον χαρακτήρα $char$, ενώ το neg_{char} αντιπροσωπεύει το Αρνητικό Διάνυσμα για τον χαρακτήρα. Το $\{pos_{char}\}_{\text{char} \in T_i}$ είναι το σύνολο των Θετικών Διανυσμάτων για τους χαρακτήρες που αναφέρονται στην τρέχουσα περιγραφή (T_i) και $\{neg_{char}\}_{\text{char} \notin T_i}$ το σύνολο των Αρνητικών Διανυσμάτων για τους χαρακτήρες που **δεν** αναφέρονται.

Εκπαίδευση Προκειμένου να ενισχύσουμε την εστίαση του μοντέλου στα Διανύσματα Χαρακτήρων, απορρίπτουμε εντελώς τις γλωσσικές περιγραφές για ένα ποσοστό δειγμάτων εκπαίδευσης σε κάθε δέσμη (batch) και κρατάμε μόνο τα Διανύσματα Χαρακτήρων ως καθοδήγηση. Κατά τα άλλα, η διαδικασία εκπαίδευσης μένη απαράλαχτη.

Συμπερασμός Κατά τον Συμπερασμό, υπολογίζουμε δύο ομάδες από πιθανότητες (logits) για να παράξουμε μια εικόνα. Η πρώτη ομάδα (ℓ_{tc}) υπολογίζεται διεκπεραιώνοντας την παραγωγική διαδικασία, υπο συνθήκη των γλωσσικών περιγραφών. Η δεύτερη ομάδα (ℓ_{char}) υπολογίζεται υπό συνθήκη των Διανυσμάτων Χαρακτήρων, αποκλειστικά. Όπως κατά την εκπαίδευση χρησιμοποιούμε Θετικα Διανύσματα για τους χαρακτήρες που εμφανίζονται στην περιγραφή και Αρνητικά για του άλλους. Οι τελικές πιθανότητες (logits) προκύπτουν ως κυρτός συνδυασμός των δύο επιμέρους ομάδων πιθανοτήτων:

$$\ell = (1 - f)\ell_{tc} + f\ell_{char}, \;\; f \in [0, 1) \tag{1.4.7}$$

Αυτός ο συνδυασμός υπολογίζεται σε κάθε βήμα της επαναληπτικής παραγωγικής διαδικασίας συμπερασμού.

Χρήση Αρνητικών Υποδείξεων Με σκοπό την περαιτέρω ενίσχυση της παρουσίας του σωστού υποσυνόλου των χαρακτήρων σε κάθε εικόνα, προτείνουμε την χρήση αρνητικής καθοδήγησης (negative prompting) κατά τον συμπερασμό. Συγκεκριμένα, εκτός από τις δύο ομάδες από logits (ℓ_{tc} and ℓ_{char}), εισάγουμε μια τρίτη ομάδα, τα $\ell_{\overline{char}}$. Για τον υπολογισμό των $\ell_{\overline{char}}$ απορρίπτουμε εντελώς τις γλωσσικές περιγραφές από την είσοδο του μετασχηματιστή, όπως κάνουμε για τα ℓ_{char}. Όμως, αντί να χρησιμοποιήσουμε Θετικά Διανύσματα για τους επιθυμητούς χαρακτήρες και Αρνητικά για τους υπόλοιπους, κάνουμε το ανάποδο. Αρνητικά Διανύσματα χρησιμοποιούνται για τους επιθυμητούς χαρακτήρες και Θετικά για τους υπόλοιπους. Δηλαδή, τα Διανύσματα Χαρακτήρων στην είσοδο είναι: $C_i = \{neg_{char}\}_{\text{char} \in T_i} \bigcup \{pos_{char}\}_{\text{char} \notin T_i}$. Κατά κάποιον τρόπο τα $\ell_{\overline{char}}$ υπολογίζονται χρησιμοποιώντας το "λογικό *συμπλήρωμα*" της εισόδου που χρησιμοποιείται για τον υπολογισμό των ℓ_{char}. Οι τελικές πιθανότητες (logits) υπολογίζονται τώρα ως εξής:

$$\ell = (1 - f)\ell_{tc} + 2f\ell_{char} - f\ell_{\overline{char}}, \;\; f \in [0, 1) \tag{1.4.8}$$

1.4.4 Επάυξηση των Δεδομέων μέσω Μεγάλων Γλώσσικών Μοντέλων

Για να προστατεύσουμε το μοντέλο ενάντια στην υπερπροσαρμογή (overfitting) πειραματιζόμαστε με την επάυξηση των κειμενικών περιγραφών. Για αυτόν τον σκοπό χρησιμοποιούμε ένα LLM, από το οποίο ζητάμε εναλλακτικές περιγραφές, δεδομένων των πραγματικών.

Κατά την εκπαίδευση, επιλέγουμε τυχαία είτε την πρωτότυπη περιγραφή είτε την εναλλακτική, για κάθε εικόνα, σε κάθε εποχή. Έτσι, παρέχουμε στο μοντέλο διαφορετικές περιγραφές για την ίδια εικόνα, σε διαφορετικές εποχές. Περιμένουμε αυτό να βοηθήσει το μοντέλο να εστιάσει στα Διανύσματα Λέξεων που είναι πιο σχετικά, κατά την παραγωγή των εικόνων, αλλά και να αποφύγει σε μεγαλύτερο βαθμό την υπερπροσαρμογή.

1.4.5 Κριτής Συμβόλων Βασισμένος στους Χαρακτήρες

Επηρεασμένοι από την ιδέα του Κριτή Συμβόλων (Token-Critic) πειραματιζόμαστε με έναν Κριτή Συμβόλων Βασιμένο στους Χαρακτήρες (Character Attentive Token Critic). Η λειτουργία του είναι ίδια με αυτήν που περιγράψαμε στην Ενότητα 1.3.3. Για την δική μας εργασία, επιλέγουμε η συνθήκη κάτω από την οποία γίνονται οι προβλέψεις να είναι Διανύσματα Χαρακτήρων. Συγκεκριμένα, ο Κριτής Συμβόλων έχει C_n διανύσματα, ένα για κάθε βασικό χαρακτήρα (C_n είναι το πλήθος των χαρακτήρων του Dataset). Εκπαιδεύουμε τον Κριτή Συμβόλων, με παραμέτρους ϕ για την ελαχιστοποίηση του:

$$\mathcal{L} = \mathbb{E}[\sum_{j=1}^{N} BCE(m_j, p_\phi(m_j|\tilde{Y}, c))] \tag{1.4.9}$$

1.4.6 Αύξηση της ευκρίνειας του Κρυφού Χώρου Χαρακτηριστικών

Εμπνευσμένοι από το MUSE[4], πειραματιζόμαστε με την χρήση ενός Μετασχηματιστή αύξησης ευκρίνειας (Super-Resolution), που λειτουργεί σε κρυφό χώρο υψηλότερης ευκρίνειας, ο οποίος παράγει αποτελέσματα με συνθήκη την χαμηλότερης ευκρίνειας έξοδο ενός Βασικό Μετασχηματιστή (Base Transformer). Για αυτόν τον σκοπό χρειάζονται δύο VQ-GAN, που λειτουργούν σε κρυφούς χώρους με διαφορετικές αναλύσεις: $m_1 \times m_1$ και $m_2 \times m_2$ ($m_1 < m_2$).

Βασικός Μετασχηματιστής Ο Βασικός Μετασχηματιστής (Base Transformer) είναι ένα MaskGST, όπως το περιγράψαμε στην ενότητα 1.4.3. Λειτουργεί στον κρυφό χώρο του VQ-GAN με ανάλυση $m_1 \times m_1$ (η χαμηλότερη ανάλυση).

Super-Resolution Μετασχηματιστής Κατά την εκπαίδευση του Super-Resolution Μετασχηματιστή έχει ολοκληρωθεί η εκπαίδευση του Βασικού Μετασχηματιστή. Ο Super-Resolution Μετασχηματιστής είναι ένα MaskGST, με ανάλυση $m_2 \times m_2$ στον κρυφό χώρο. Επίσης υπάρχει μια τροποποίηση στην είσοδο.

Δεδομένων των οπτικών συμβόλων υψηλής ανάλυσης (HR) $Z = \{Z_1, Z_2, ..., Z_n\}$ ($Z_i \in \mathbb{R}^{m_2 \times m_2 \times d}$), που κωδικοποιούνται μέσω του αντίστοιχου VQ-GAN, των γλωσσικώσ συμβόλων $T = \{T_1, T_2, ..., T_n\}$ ($T_i \in \mathbb{R}^{l \times d}$) και των οπτικών συμβόλων χαμηλής ανάλυσης (LR), $Z^{LR} = \{Z_1^{LR}, Z_2^{LR}, ..., Z_n^{LR}\}$ ($Z_i^{LR} \in \mathbb{R}^{m_1 \times m_1 \times d}$), σχηματίζουμε την είσοδο του Super-Res Μετασχηματιστή όπως φαίνεται στην Εικόνα 1.4.10. Τα οπτικά σύμβολα $Z \in \mathbb{R}^{n \times m_2 \times m_2 \times d}$ ισοπεδώνονται σε μια ακολουθία $Z' \in \mathbb{R}^{n \times (m_2 \cdot m_2) \times d}$. Στην συνέχεια, τους εφαρμόζεται τυχαία μάσκα, όπως στο *MaskGIT*, ώστε να μετατραούν στα $\bar{Z} \in \mathbb{R}^{n \times (m_2 \cdot m_2) \times d}$. Τα χαμηλής ανάλυσης οπτικά σύμβολα ισοπεδώνονται κατά των ίδιο τρόπο και παιρνάνε από μια σειρά Αυτο-Στρωμάτων (όπως γίνεται και στο MUSE[4]). Τέλος, τα HR οπτικά σύμβολα που τους έχει εφαρμοστεί μάσκα, τα γλωσσικά σύμβολα και τα παραχθέντα LR οπτικά σύμβολα συνενώνονται για να σχηματίσουν το $Input_i = (\bar{Z}_i; T_i; Z_i^{LR})$. Η είσοδος του Μετασχηματιστή μπορεί να γραφτεί ως εξής:

$$Input = \{Input_1, ..., Input_n\} \in \mathbb{R}^{n \times (m_2 \cdot m_2 + l + m_1 \cdot m_1) \times d} \tag{1.4.10}$$

όπου l είναι το μήκος των γλωσσικών συμβόλων και d η κρυφή διάσταση του Μετασχηματιστή.

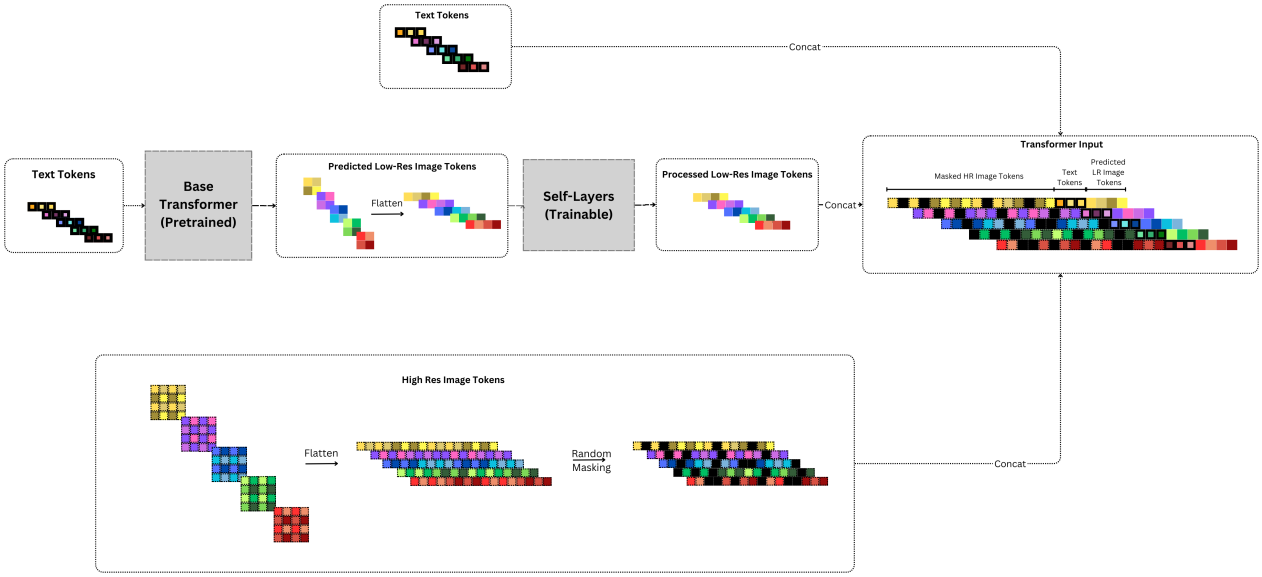


Figure 1.4.10: Η είσοδος του Μοντέλου Super-ResThe σχηματίζεται συνενώνοντας τα High-Resolution(HR) οπτικά σύμβολα με τα γλωσσικύ σύμβολα και τα Low-Resolution(LR) οπτικά σύμβολα, που έχει προβλέψει το βασικό μοντέλο

1.4.7 Απόπλεξη Χαρακτηριστικών στον Κρυφό Χώρο

Με σκοπό την βελτίωση της παραγωγής των χαρακτήρων προτείνουμε μια τεχνική που επιχειρεί να αποπλέξει τα χαρακτηριστικά που αντιστοιχούν σε χαρακτήρες από τα υπόλοιπα, στον κρυφό χώρο. Η τεχνική μας περιλαμβάνει την τροποποίηση του VQ-GAN για να προστεθεί μια επιπλέον διακριτή βιβλιοθήκη διανυσμάτων.

Κωδικοποιητής και Αποκωδικοποιητής του VQ-GAN

Όπως δείχνουν οι εικόνες 1.4.11a and 1.4.11b, ο Κωδικοποιητής και ο Αποκωδικοποιητής παραμένουν ίδιοι, όπως στο παραδοσιακό VQ-GAN.

Κβαντισμός με δύο βιβλιοθήκες διανυσμάτων Κρυφού Χώρου

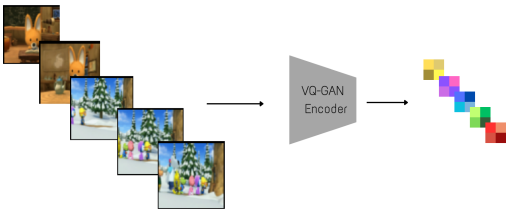
Αντί να χρησιμοποιήσουμε μία βιβλιοθήκη διανυσμάτων στον Κρυφό Χώρο, όπως στο VQ-GAN, χρησιμοποιούμε δύο: $e^{background} \in \mathbb{R}^{K \times D}$ και $e^{char} \in \mathbb{R}^{K \times D}$, όπου με K συμβολίζεται ο αριθμός των διακριτών διανυσμάτων σε κάθε βιβλιοθήκη και με D η διαστατικότητα των διανυσμάτων. Η διαίσθηση πίσω από αυτή την ιδέα είναι ότι το $e^{background}$ κωδικοποιεί χαρακτηριστικά (features) που αντιστοιχούν στο φόντο των εικόνων και το e^{char} χαρακτηριστικά που αντιστοιχούν στους χαρακτήρες. Η διαδικασία κβαντισμού φαίνεται στην Εικόνα 1.4.11c.

Μία εικόνα $x \in \mathbb{R}^{3 \times N \times N}$ περνάει πρώτα από τον Κωδικοποιητή ώστε να μετατραπεί στο $z_0 \in \mathbb{R}^{D \times \frac{N}{f} \times \frac{N}{f}}$. Εφαρμόζουμε ένα D-διάστατο φίλτρο $f_{background}$ στο z_0 για να πάρουμε το $z_1 = f_{background}(z_0)$. Στην συνέχει κβαντίζουμε το z_1 σύμφωνα με το $e^{background}$, αντικαθιστόντας καθένα από τα $(\frac{N}{f} \times \frac{N}{f})$ διανύσματα (καθένα έχει D διαστάσεις) με το κοντινότερό του στην βιβλιοθήκη. Το αποτέλεσμα του κβαντισμού είναι το $z_{background} \in \mathbb{R}^{D \times \frac{N}{f} \times \frac{N}{f}}$.

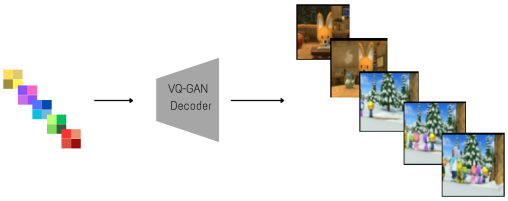
Παρόμοια, χρησιμοποιούμαι ένα άλλο D-διάστατο φίλτρο f_{char} για να πάρουμε το $z_2 = f_{char}(z_0)$. Μετά κβαντίζουμε το z_2 με βάση το e^{char} και παίρνουμε το $z_{char} \in \mathbb{R}^{D \times \frac{N}{f} \times \frac{N}{f}}$.

Συνδυάζουμε τα z_{char} και $z_{background}$ χρησιμοποιώντας μια Μάσκα Φόντου (background mask). Έστω $M \in \mathbb{R}^{\frac{N}{f} \times \frac{N}{f}}$ η Μάσκα Φόντου με:

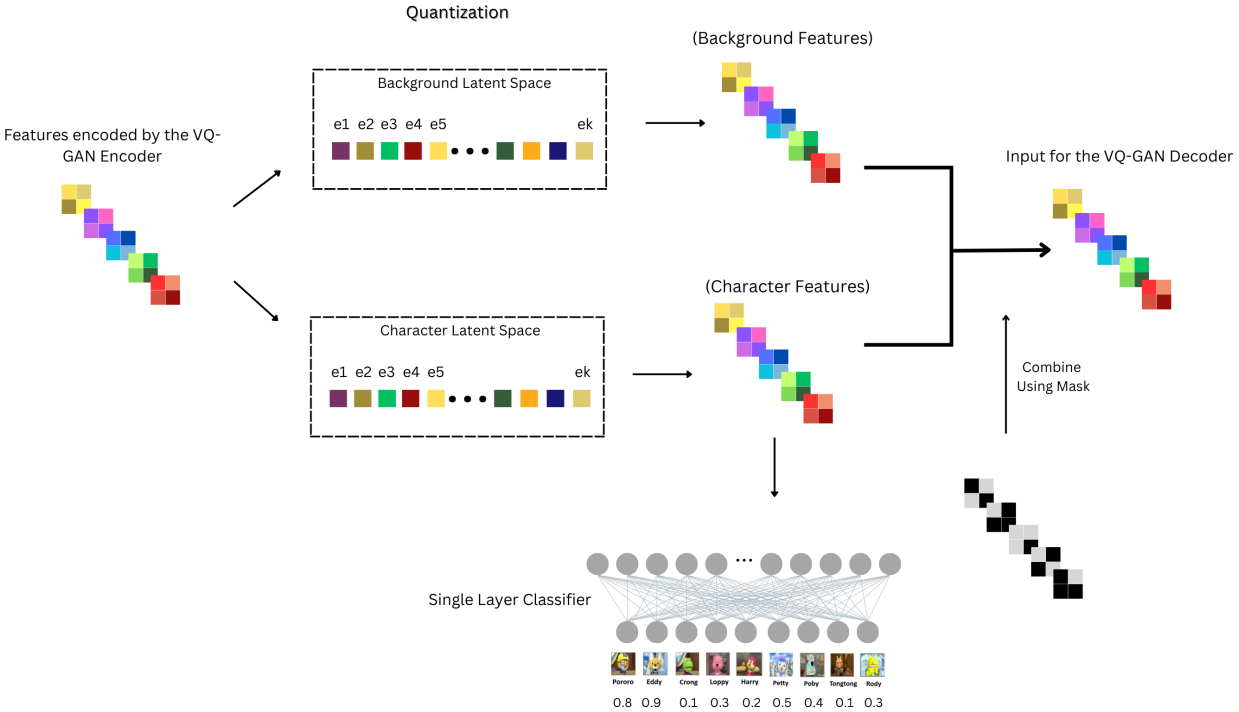
$$M_{i,j}^{background} = \begin{cases} 0 & \text{if the (i,j) region in the original image belongs to the foreground} \\ 1 & \text{if the (i,j) region in the original image belongs to the background} \end{cases} \tag{1.4.11}$$



(a) Ο Κωδικοποιητής του VQ-GAN μένει ίδιος



(b) Ο Αποκωδικοποιητής του VQ-GAN μένει, επίσης ίδιος



(c) Τροποποιούμε το στάδιο του κβαντισμού, προσπαθώντας να πετύχουμε απόπλεξη χαρακτηριστικών

Figure 1.4.11: Τροποποιημένο VQ-GAN για απόπλεξη χαρακτηριστικών

Όπου περιοχές μια εικόνας με χαρακτήρες θεωρείται ότι ανήκουν στο προσκήνιο (foreground), ενώ περιοχές χωρίς χαρακτήρες ανήκουν στο φόντο (background). Προβάλουμε το $M^{background}$ από $M^{background} \in \mathbb{R}^{\frac{N}{f} \times \frac{N}{f}}$ σε $\hat{M}^{background} \in \mathbb{R}^{D \times \frac{N}{f} \times \frac{N}{f}}$. Η τελική αναπαράσταση της εικόνας στον Κρυφό Χώρο σχηματίζεται ως εξής:

$$z = z_{background} \odot \hat{M}^{background} + z_{char} \odot \overline{\hat{M}^{background}} \tag{1.4.12}$$

Δηλαδή, για τις περιοχές που ανήκουν στο φόντο, χρησιμοποιούμε τα διανύσματα του $z_{background}$, ενώ για περιοχές που ανήκουν στο προσκήνιο τα διανύσματα του z_{char}. Το z μπορεί αν δοθεί στον Αποκωδικοποιητή για να παράξει την ανακατασκευασμένη εικόνα.

Όπως φαίνεται στην Εικόνα 1.4.11c, το z_{char} περνά και από ενα Νευρωνικό Δίκτυο ενός επιπέδου. Αυτό το δίκτυο παίρνει την ισοπεδωμένη έκδοση του z_{char}, $\tilde{z}_{char} \in \mathbb{R}^{D \cdot \frac{N}{f} \cdot \frac{N}{f}}$ και δίνει τις πιθανότητες κλάσεων για όλους τους βασικούε χαρακτήρες του Dataset (9 για το Pororo-SV). Εκπαιδεύουμε το μικρό αυτό δίκτυο με μία συνθήκη ταξινόμησης πολλαπλών ετικετών (multi-label classification loss), χρησιμοποιώντας τις αναφορές στους χαρακτήρες στις γλωσσικές περιγραφές ως ground-truth ετικέτες. Ο σκοπός αυτού του δικτύου είναι να δώσει επιπλέον ερέθισμα στην e^{char} κατά την εκπαίδευση. Επιλέγουμε ένα ρηχό δίκτυο, ελπίζοντας ότι αυτό θα εξαναγκάσει την βιβλιοθήκη διανυσμάτων να αποπλέξει (disentangle) τα κρυφά χαρακτηριστικά που κωδικοποιούν διαφορετικούς χαρακτήρες.

Διαχωρισμό Προσκηνίου-Φόντου

Για τον διαχωρισμό Προσκηνίου-Φόντου χρησιμοποιούμε το GradCAM [31], με τον τρόπο που προτείνεται στο [5].

Τροποποιήσεις στον Μετασχηματιστή

Ο Μετασχηματιστής που χρησιμοποιούμε σε συνδυασμό με αυτή την μέθοδο είναι πολύ παρόμοιος με αυτούς που έχουμε συζητήση ήδη, με κάποιες μικρές τροποποιήσεις.

Εκπαίδευση Όπως είπαμε στην Ενότητα 1.4.3, αφού απορρίψουμε το κειμενικό κομμάτι από την έξοδο του Μετασχηματιστή παίρνουμε το εξής αποτέλεσμα:

$$\overline{Output} = \{\overline{Output_1}, ..., \overline{Output_n}\} \in \mathbb{R}^{n \times (m \cdot m) \times d} \tag{1.4.13}$$

Μέχρι τώρα χρησιμοποιούσαμε ένα μοναδικό γραμμικό μετασχηματισμό, για να μετατρέψει τις εξόδους σε πιθανότητες (logits), που ήταν αρκετό για να προβλέψουμε τα οπτικά σύμβολα στον κρυφό χώρο του πρωτότυπου VQ-GAN.

Ωστόσο, χρησιμοποιώντας το τροποποιημένο VQ-GAN, με δύο βιβλιοθήκες, χρειάζεται να προβλέψουμε της πιθανότητες για τους χαρακτήρες (Character Logits), τις πιθανότητες για το φόντο (Background logits) και την Μάσκα Φόντου (Background Mask)

Προκειμένου να το πετύχουμε αυτό χρησιμοποιούμε τρεις γραμμικούς μετασχηματισμούς: $\mathbf{To\_Char\_Logits}, \mathbf{To\_Background\_Logits}, \mathbf{To\_Mask}$. Όπου:

$$\begin{aligned} y^{char} &= \mathbf{To\_Char\_Logits}(\overline{Output}) \in \mathbb{R}^{n \times (m \cdot m + l) \times K} \\ y^{background} &= \mathbf{To\_Background\_Logits}(\overline{Output}) \in \mathbb{R}^{n \times (m \cdot m + l) \times K} \\ y^{mask} &= \mathbf{To\_Mask}(\overline{Output}) \in \mathbb{R}^{n \times (m \cdot m + l) \times 1} \end{aligned} \tag{1.4.14}$$

Η συνθήκη εκπαίδευσης είναι:

$$\mathcal{L} = \mathcal{L}_{char} + \mathcal{L}_{background} + \mathcal{L}_{mask}, \tag{1.4.15}$$

όπου:

$$\begin{aligned}
\mathcal{L}_{char} &= -\mathbb{E}[\sum_{\forall i \in [1,N], m_i=1, m_i^{background}=0} \mathbf{log} p(y_i^{char} | Y_{\bar{M}})] \\
\mathcal{L}_{background} &= -\mathbb{E}[\sum_{\forall i \in [1,N], m_i=1, m_i^{background}=1} \mathbf{log} p(y_i^{background} | Y_{\bar{M}})] \\
\mathcal{L}_{mask} &= \mathcal{L}_{CE}(y^{mask}, M^{background})
\end{aligned} \tag{1.4.16}$$

Το Y τους δείκτες (indices) για τα διανύσματα στο z, όπου το z δίνεται από την εξίσωση Equation 1.4.12, ενώ $Y_{\bar{M}}$ είναι το ίδιο μετά την εφαρμογή μασκών, σύμφωνα με το πρόγραμμα του MaskGIT. Σημειώνουμε ότι η τυχαία μάσκα M είναι διαφορετική από την Μάσκα Φόντου, $M^{background}$. $\mathcal{L}_{char}$ είναι η Συνθήκη τύπου-MaskGIT για τις περιοχές με Χαρακτήρες στις εικόνες. $\mathcal{L}_{background}$ είναι η αντιστοιχη συνθήκη για τις περιοχές φόντου. $\mathcal{L}_{mask}$ είναι η ετεροεντροπία μεταξύ της προβλεπόμενης και της ground-truth Μάσκας Φόντου.

Κατά τα άλλα το μοντέλο μένει ίδιο με το αρχικό μας MaskGST.

Συμπερασμός Κατά τον συμπερασμό χρειάζεται να προβλεψουμε την Μάσκα Φόντου, τα logits των Χαρακτήρων και τα logits του Φόντου. Ξεκινάμε παράγοντας την Μάσκα Φόντου. Αυτό το κάνουμε με ένα μοναδικό πέρασμα από τον Μετασχηματιστή. Για αυτό το πέρασμα, όλα τα οπτικά σύμβολα στην είσοδο είναι αντικατεστημένα με Μάσκα. Παίρνουμε την έξοδο όπως περιγράφεται από την εξίσωση 1.4.13. Τέλος, σχηματίζουμε το $\sigma(\mathbf{To\_Mask}(\overline{Output})) \in \mathbb{R}^{n \times m \times m}$, όπου το σύμβολο $\sigma(.)$ αναπαριστά την σιγμοϊδή (sigmoid) συνάρτηση. Η Μάσκα Φόντου προβλέπεται ως εξής:

$$\tilde{M}_{i,j,k}^{background} = \begin{cases} 1 & \text{if } \sigma(\mathbf{To\_Mask}(\overline{Output}))_{i,j,k} \geq 0.5 \\ 0 & \text{if } \sigma(\mathbf{To\_Mask}(\overline{Output}))_{i,j,k} < 0.5 \end{cases} \tag{1.4.17}$$

Αφού προβλέψουμε την Μάσκα Φόντου, κάνουμε Επαναληπτικό, Παράλληλο Συμπερασμό, όμοια με το MaskGIT. Η μόνη διαφορά ειναι ότι σε κάθε βήμα προβλέπουμε τα logits των Χαρακτήρων και τα logits του φόντου. Σχηματίζουμε τα ενιαία logits σε αυτό το βήμα σαν:

$$Logits_{i,j,k} = \begin{cases} Logits_{i,j,k}^{Background} & \text{if } \tilde{M}_{i,j,k}^{background} = 1 \\ Logits_{i,j,k}^{Char} & \text{if } \tilde{M}_{i,j,k}^{background} = 0 \end{cases} \tag{1.4.18}$$

1.5 Πειραματικό Μέρος

1.5.1 Οργάνωση των Πειραμάτων

Κώδικας

Όλος ο κώδικας μας έχει αναπτυχθεί σε PyTorch. Ο κώδικας για το MaskGIT βασίστηκε σε μια υλοποίση ανοιχτού κώδικα για το MUSE[1]. Για το VQ-GAN χρησιμοποιούμε την πρωτότυπη υλοποίηση από το *Taming Transformers* [8], που διατίθεται στο github[2]. Για τον κωδιποιητή BPE χρησιμοποιήσαμε την κλάση *Tokenizer* που παρέχεται από το Hugging Face[3].

Περιβάλλον Εκπαίδευσης

Όλα μας τα πειράματα πραγματοποιήθηκαν στην δομή ARIS, του GRNET. Συγκεκριμένα, για κάθε πείραμα χρησιμοποιήσαμε μία μοναδική κάρτα γραφικών NVIDIA V100-16GB.

[1] https://github.com/lucidrains/muse-maskgit-pytorch
[2] https://github.com/CompVis/taming-transformers
[3] https://huggingface.co/learn/nlp-course/chapter6

Σύνολο Δεδομένων

Pororo-SV Το βασικό Σύνολο Δεδομένων (Dataset) για την Οπτικοποίηση Ιστορίας, στην βιβλιογραφία είναι το Pororo-SV, που προτάθηκε στο [18]. Βασίζεται στην παιδική σειρά κινουμένων σχεδίων "Pororo the Little Penguin". Περιέχει 15,336 ιστορίες συνολικά. Κάθε ιστορία αποτελείται από 5 εικόνες με τις αντίστοιχες περιγραφές. Ακολουθώντας τους [20], υϊοθετούμε ένα χωρισμό με 10191/2334/2208 ιστορίες train/validate/test (εκπαίδευσης/επικύρωσης/εξέτασης). Το Dataset περιέχει 9 επαναλαμβανόμενους Χαρακτήρες.

Μετρικές για την Οπτικοποίηση Ιστορίας

Ακολουθώντας τους [20], υϊοθετούμε τις πιο διαδομένες μετρικές για την Οπτικοποίηση Ιστορίας. Είναι οι εξής:

- **FID** που συκρίνει την κατανομή την χαρακτηριστικών (features) μεταξύ των ground-truth και των παραγόμενων εικόνων. Για την εξαγωγή των χαρακτηριστικών χρησιμοποιείται ένα προεκπαιδευμένο Inception v3.
- **Character-F1** που ποσοτικοποιεί κατά πόσο το μοντέλο παράγει τους επιθυμητούς Χαρακτήρες στις εικόνες. Χρησιμοποιούμε έναν προεκπαιδευμένο ταξινομητή που αναγνωρίζει την παρουσία των χαρακτήρων στις εικόνες και υπολογίζουμε το F1-Score ανάμεσα στα αποτελέσματα τους ταξινομητή για τις εικόνες μας και στις ground-truth αναφορές των περιγραφών.
- **Character-Accuracy** που λειγουργεί όπως το Character-F1, απλώς υπολογίζει την ακρίβεια (Accuracy).
- **BLEU-2/3** που χρησιμοποιεί ένα μοντέλο υποτιτλισμού (captioning) βίντεο για να παράξει μια γλωσσική περιγραφή για κάθε παραγόμενη ιστορία (πεντάδα εικόνων) και για την αντίστοιχη πραγματική (ground-truch) πεντάδα. Στην συνέχεια συγκρίνει τις περιγραφές με βάση την μετρική BLEU.

1.5.2 Πειράματα Αρχιτεκτονικής

Κωδικοποίηση Εικόνων: VQ-GAN

Στα αρχικά στάδια πειραματισμού δοκιμάσαμε VQ-GANs me παράγοντες συμπίεσης: $f = 4$, $f = 8$ and $f = 16$, που αντιστοιχούν σε αναπαραστάσεις στον κρυφό χώρο με ανάλυση 16×16, 8×8 and 4×4 respectively (όλες οι εικόνες έχουν ανάλυση 64×64). Οι διαστάσεις της βιβλιοθήκης διανυσμάτων Κρυφού Χώρου που χρησιμοποιούμε είναι 128×256, Δηλαδή 128 διακριτά διανύσματα με διάσταση 256 το καθένα. Όλα τα μοντέλα έχουν $\sim 35M$ παραμέτρους.

Καταλήξαμε ότι μεταξύ των τριών περιπτώσεων αυτή που καλύτερα συνδυαζόταν με τους Μετασχηματιστές ήταν αυτή με $f = 8$ (ανάλυση 8×8 για την κρυφή αναπαράσταση). Η υπεροχή αυτής της περίπτωσης σε σχέση με τις άλλες ήταν σημαντική σε όλες τις μετρικές και ιδίως στο FID. Υποθέτουμε ότι αυτό έχει να κάνει με την λεπτομέρεια των κρυφών αναπαραστάσεων, ανάλογα με την συμπίεση που υπόκεινται. Πράγματι, όσο μεγαλύτερος είναι ο παράγοντας συμπίεσης, τόσο πιο "χοντροκομμένα" θα είναι τα χαρακτηριστικά (features) που κωδικοποιούν τα διακριτά οπτικά σύμβολα του VQ-GAN, αφού μεγαλύτερο μέρος της εικόνας θα πρέπει να κωδικοποιηθεί από ένα μόνο σύμβολο. Υποθέτουμε ότι το VQ-GAN με $f = 8$ πετυχαίνει την ιδανική ισορροπία όσων αφορά τα οπτικά σύμβολα, ώστε να είναι κατάλληλα για να τα προβλέψει ο Μετασχηματιστής από κείμενο. Από εδώ και πέρα, στα πειράματα χρησιμοποιούμε VQ-GAN με ανάλυση 8×8 στην κρυφή αναπαράσταση.

MaskGST

Όπως αναφέρηκε στην Ενότητα 1.4.3 το MaskGST αποτελείται από δύο Πλήρη Στρώματα, ακολουθούμενα από κάποια Αυτο-Στρώματα. Για αυτό το πείραμα, επιλέγουμε 4 Αυτο-Στρώματα (6 συνολικά στρώματα για τον Μετασχηματιστή). Το μοντέλο είναι όμοιο με το παράδειγμα της Εικόνας 1.4.7. Η κρυφή διάσταση είναι $d = 1024$. Το μέγεθος του λεξιλογίου κατά την BPE-κωδικοποίηση $n_{vocab} = 2500$. Ο αριθμός των κεφαλών Προσοχής είναι $n_{heads} = 8$. Ο μετασχηματιστής έχει $70M$ παραμετρους και το συνολικό μοντέλο, μαζί με το VQ-GAN έχει 105M parameters. Εκπαιδεύουμε για 200 Εποχές με ρυθμό μάθησης $lr = 5e-3$. Όπως δείχνει ο πίνακας 9.6, το MaskGST ξεπερνά προηγούμενες αρχιτεκτονικές GAN σε όλες τις μετρικές. Επίσης πλησιάζει τις προηγούμενες δουλειές με Μετασχηματιστές ξεπερνώντας τες σε Character Accuracy.

Χρησιμοποιούμε αυτό το μοντέλο σαν την γραμμή αναφοράς μας (baseline). Όλα τα επόμενα πειράματα χτίζουν πάνω σε αυτή την ιδέα με διαφορετικούς τρόπους. Σημειώνουμε, επίσης ότι αν δεν αναφέρεται διαφορετικά οι

υπερπαράμετροι παραμένουν ίδιες και στα επόμενα πειράματα.

MaskGST-SV

Στην Ενότητα 1.4.3 περιγράψαμε το MaskGST-SV ως ένα μοντέλο που ξεκινά με δύο Πλήρη Στρώματα, που προαιρετικά ακολουθούνται από κάποια Αυτο-Στρώματα, τα οποία ακολουθούνται απο κάποια SV-Στρώματα, που, τέλος ακολουθούνται προαιρετικά από κάποια ακόμα Αυτο-Στρώματα. Πειραματιζόμαστε με τρεις διαφορετικές διατάξεις που φαίνονται στην Εικόνα 9.2.1.

Η διάταξη (a) (Εικόνα 9.2.1a) αποτελείται από 2 Πλήρη Στρώματα, ακολουθούμενα από 2 SV-Στρώματα, ακολουθούμενα από 4 Αυτο-Στρώματα. Η διάταξη (b) (Figure 9.2.1b) αποτελείται από 2 Πλήρη Στρώματα, ακολουθούμενα από 2 Αυτο Στρώματα, ακολουθούμενα από 2 SV-Στρώματα, ακολουθούμενα από 2 Αυτο-Στρώματα. Τέλος, η διάταξη (c) (Figure 9.2.1c) βάζει στην σειρά 2 Πλήρη Στρώματα, 4 Αυτο-Στρώματα, και 2 SV-Στρώματα. Είναι φανερό ότι και οι τρεις εναλλακτικές χρησιμοποιούν 2 Πλήρη Στρώματα, 4 Αυτο-Στρώματα και 2 SV-Στρώματα. Αλλάζει η θέση των SV-Στρωμάτων. Ξεκινώντας από την πρώτη προς της τρίτη διάταξη, τα SV-Στρώματα μετακινούνται από την αρχή προς το τέλος της παραγωγικής διαδικασίας.

Οι λοιπές υπερπαράμετροι μένουν ίδιες με του MaskGST ($d = 1024$, $n_{vocab} = 2500$, $n_{heads} = 8$. $lr = 5e - 3$, $n_{epochs} = 200$).

Τα πειράματά μας δείχνουν ότι οι διατάξεις (a) και (b) βελτιώνουν τα αποτελέσματα σε σχέση με το συμβατικό MaskGST. Αντίθετα, η διάταξη (c) δίνει σημαντικά χειρότερα αποτελέσματα, ακόμα και από το MaskGST. Λεπτομερή αποτελέσματα φαίνονται στον Πίνακα 9.1. Συμπεραίνουμε ότι η χρήση SV-Στρωμάτων μπορεί να ενισχύσει την απόδοση του μοντέλου, αλλά παίζει σημαντικό ρόλο πού θα τοποθετηθούν στην παραγωγική διαδικασία. Τα αποτελέσματά μας υποδεικνύουν ότι δεν πρέπει να τοποθετηθούν στο τέλος του Μετασχηματιστή.

Το T5-XXL ως κωδικοποιητής κειμένου

Για να πειραματιστούμε με το T5-XXL[24] ως κωδικοποιητή κειμένου χρησιμοποιούμε το MaskGST, χωρίς τα διανύσματα λέξεων (text embeddings). Αντί αυτών, παίρνουμε μία κωδικοποίηση για κάθε γλωσσική περιγραφή περνώντας την μέσα από ένα προ-εκπαιδευμένο μοντέλο T5-XXL και χρησιμοποιώντας τις τελευταίες κρυφές καταστάσεις αντί διανυσμάτων λέξεων για τον Μετασχηματιστή μας. Όπως φαίνεται στον Πίνακα 9.3 (MaskGST w/ T5-XXL), αυτή η προσέγγιση δεν βελτιώνει σημαντικά καμία μετρική σε σχέση με το συμβατικό MaskGST. Πιστεύουμε ότι αυτό οφείλεται στο γεγονός ότι το T5-XXL δεν έχει εκπαιδευτεί στο σώμα κειμένων (corpus) των περιγραφών του Dataset που χρησιμοποιούμε. Αυτό το corpus έχει όμως ιδιαιτερότητες, όπως τα ονόματα των Χαρακτήρων που δεν αποτελούν πραγματικές Αγγλικές λέξεις (π.χ. Pororo, Crong κλπ). Υποθέτοντας, ότι για τέτοιες λέξεις το T5 δεν καταφέρνει να συνθέσει εκφραστικές αναπαραστάσεις, είναι λογικό το ότι δεν είναι καλά τα αποτελέσματα, αφού αυτές οι λέξεις είναι οι σημαντικότερες για την σύνθεση των εικόνων, στην περίπτωσή μας.

Επάυξηση Γλωσσικών Δεδομένων μέσω του ChatGPT

Χρησιμοποιούμε το ChatGPT 3.5 μέσω του API που παρέχει η OpenAI για να επαυξήσουμε τις γλωσσικές περιγραφές του Dataset. Για καθε ιστορία (σειρά από 5 περιγραφές), δίνουμε στο ChatGPT την περιγραφή του ρόλου του ως βοηθού για την επαύξηση δεδομένων, ακολουθούμενη από τις 5 περιγραφές. Το ακριβές μήνυμα ρόλου που δίνουμε στο μοντέλο (για το Pororo-SV) παρατίθεται στην Ενότητα 9.2.5. Οι πρωτότυπες περιγραφές της ιστορίας δίνονται στο μοντέλο στην παρακάτω μορφή:

```
1. {περιγραφή 1}
2. {περιγραφή 2}
3. {περιγραφή 3}
4. {περιγραφή 4}
5. {περιγραφή 5}
```

Το μοντέλο που χρησιμοποιούμε για το πείραμα είναι όμοιο με το MaskGST του αρχικού πειράματος. Η μόνη διαφορά είναι ότι κατά την εκπαίδευση επιλέγουμε τυχαία ανάμεσα στην πρωτότυπη περιγραφή και αυτή που έχει παραχθεί απο το ChatGPT για κάθε δείγμα εκπαίδευσης, σε κάθε εποχή. Κατά τον συμπερασμό χρησιμοποιούμε μόνο τις πρωτότυπες περιγραφές.

Όπως φαίνεται στον πίνακα 9.3, αυτό το πείραμα (MaskGST w/ aug. captions) πετυχαίνει βελτιωμένα αποτελέσματα σε σχέση με το συμβατικό MaskGST σε όλες τις μετρικές, με εξαίρεση τις μετρικές BLEU, αν και εκεί δεν είναι μεγάλη η διαφορά. Υποθέτουμε ότι αυτό το αποτέλεσμα επιβεβαιώνει ότι η επαύξηση των κειμενικών δεδομένων εκπαίδευσης προστατεύει το μοντέλο από την υπερ-προσαρμογή (overfitting) και το βοηθάει να εστιάσει περισσότερο σε σημαντικές έννοιες (π.χ. ονόματα Χαρακτήρων), παρέχοντας εναλλακτικές περιγραφές για κάθε εικόνα.

Καθοδήγηση Χαρακτήρων

Για να διεξάγουμε αυτό το πείραμα χρησιμοποιούμε το MaskGST (όπως στο αρχικό πείραμα) και προσθέτουμε $2C_n$ Διανύσματα Χαρακτήρων (ένα Θετικό και ένα Αρνητικό για κάθε Χαρακτήρα), όπως αναλύσαμε στην Ενότητα 1.4.3.

Κατά την εκπαίδευση, θέτουμε την πιθανότητα απόρριψης της κειμενικής περιγραφής σε 20%, δηλαδή απορρίπτουμε τελείως την περιγραφή στο 20% των δειγμάτων εκπαίδευσης, σε κάθε εποχή και τα αντικαθιστούμε με ένα διάνυσμα `[NULL]`. Για αυτά τα δείγματα, ο Μετασχηματιστής προβλέπει τα οπτικά σύμβολα στα οποία έχει εφαρμοστεί μάσκα βασιζόμενος μόνο στην Αυτο-Προσοχή μεταξύ οπτικών συμβόλων και στα Διανύσματα Χαρακτήρων. Το υπόλοιπο 80% των δειγμάτων εκπαίδευσης, χρησιμοποιούνται τόσο οι γλωσσικές περιγραφές, όσο και τα Διανύσματα Χαρακτήρων ως συνθήκες.

Κατά τον συμπερασμό σχηματίζουμε logits, υπο συνθήκη κειμένου, ℓ_{tc} και logits, υπό συνθήκη Διανυσμάτων Χαρακτήρων ℓ_{char}. Τα τελικά logits δίνονται από τον τύπο $\ell = (1-f)\ell_{tc} + f\ell_{char}$. Χρησιμοποιούμαι $f = 0.2$, που είανι συνεπές με την πιθανότητα απόρριψης γλωσσικών περιγραφών στην εκπαίδευση.

Τα αποτελέσματα για αυτό το πείραμα φαίνονται στον πίνακα 9.3 (MaskGST-CG$_+$). Έχουμε σημαντική βελτίωση των αποτελεσμάτων σε σχέση με το αρχικό πείραμα. Η υπόθεσή μας ότι η χρήση ξεχωριστών logits για τους Χαρακτήρες θα βελτιώση την παραγωγή των Χαρακτήρων επιβεβαιώνεται από την σημαντική βελτίωση των Char-F1 και Char-Acc. Επιπρόσθετα, παρατηρούμε βελτίωση και στις υπόλοιπες μετρικές (FID και BLEU-score).

Χρήση Αρνητικών Υποδείξεων

Σε αυτό το πείραμα χρησιμοποιούμε το ίδιο μοντέλο με το προηγούμενο (Καθοδήγηση Χαρακτήρων). Επομένως, δεν αλλάζει τίποτα όσον αφορά την αρχιτεκτονική και την εκπαίδευση. Αυτό που τροποποιείται είναι το σχήμα συμπερασμού. Σε κάθε επανάληψη της διαδικασίας συμπερασμού, υπολογίζουμε τα logits, υπό συνθήκη γλωσσικών περιγραφών και το Αρνητικά logits Χαρακτήρων ($\ell_{\overline{char}}$), όπως περιγράφουμε στην Ενότητα 1.4.3. Τα τελικά logits (ℓ) σε κάθε βήμα υπολογίζονται ως εξής: $\ell = (1+f)\ell_{tc} - f\ell_{\overline{char}}$.

Τα αποτελέσματα αυτού του πειράματος φαίνονται στον Πίνακα 9.3 (MaskGST-CG$_-$). Είναι φανερό ότι η Χρήση Αρνητικών Υποδείξεων έχει θετικά αποτελέσματα σε όλες τις μετρικές.

Ένας πιο πλήρης τρόπος να χρησιμοποιήσουμε την ιδεά των Αρνητικών Υποδείξεων είναι σε συνδυασμό με (Θετική) Καθοδήγηση Χαρακτήρων, με σκοπό να σπρώξουμε τα logits προς τους επιθυμητούς Χαρακτήρες και μακριά από ανεπιθύμητους, σε κάθε εικόνα. Για να το πετύχουμε αυτό σχηματίζουμε τα logits (ℓ) σε κάθε επανάληψη του συμπερασμού σαν συνδυασμό τριών ανεξάρτητων ομάδων από logits: ℓ_{tc}, ℓ_{char} και $\ell_{\overline{char}}$. Χρησιμοποιούμε τον τύπο:

$$\ell = (1-f)\ell_{tc} + 2f\ell_{char} - f\ell_{\overline{char}} \tag{1.5.1}$$

όπου $f = 0.2$. Σημειώνουμε ότι το άθροισμα των συντελεστών για τα logits είναι $(1-f) + 2f - f = 1$.

Τα αποτελέσματα αυτής της μεθόδου είναι στον Πίνακα 9.3 (MaskGST-CG$_\pm$). Παρατηρούμε ότι ο συνδυασμός της Καθοδήγησης Χαρακτήρων με την Αρνητική Καθοδήγηση είναι πράγματι ωφέλιμος. Φαίνεται ότι οι δύο μέθοδοι ενισχύουν η μία την άλλη και φέρνουν φέρνουν όλες τις μετρικές στο καλύτερο επίπεδο σε σχέση με όλα τα προηγούμενα πειράματα.

Κριτής Συμβόλων Βασισμένος στους Χαρακτήρες

Όπως περιγράψαμε στην Ενότητα 1.4.5, ο Κριτής Συμβόλων (Token Critic) είναι ένας βοηθητικός Μετασχηματιστής που χρησιμοποιεί Αυτο-Προσοχή μεταξύ των οπτικών συμβόλων και Ετερο-Προσοχή με εκπαιδευόμενα

Διανύσματα Χαρακτήρων για να βγάλει ως έξοδο Βαθμούς Εμπιστοσύνης για τα Οπτικά Σύμβολα, σε κάθε επανάληψη την διαδικασίας συμπερασμού. Ο Μετασχηματιστής που χρησιμοποιούμε αποτελείται απο 4 διαδοχικά Πλήρη Στρώματα (φαίνεται στην Εικόνα 9.2.2). Παίρνει σαν είσοδο την έξοδο του MaskGST και βγάζει στην έξοδο έναν αριθμό μεταξύ 0 και 1 για κάθε οπτικό σύμβολο (ο βαθμός εμπιστοσύνης του). Εκπαιδεύουμε έναν Κριτή Συμβόλων με διάσταση $d = 512$ και αριθμό κεφαλών $n_{heads} = 8$. Ο παραγωγικός Μετασχηματιστής που χρησιμοποιούμε είανι ένα MaskGST όμοιο με εκείνο του αρχικού πειράματος. Το ολοκληρωμένο μοντέλο (VQ-GAN, MaskGST και Κριτής Συμβόλων) έχει $139M$ παραμέτρους.

Τα αποτελέσματα του πειράματος φαίνονται στον πίνακα 9.3 (MaskGST w/ Char-Attn T.C.). Βλέπουμε ουσιαστική πτώση των επιδόσεων στις μετρικές, από την χρήση του Κριτή Συμβόλων σε σχέση με το αρχικό πείραμα. Μία πιθανή εξήγηση είναι ότι η χρήση Διανυσμάτων Χαρακτήρων ως μοναδικό συγκείμενο (context) για τον Κριτή Συμβόλων είναι ανεπαρκής για να προβλέψει βαθμούς εμπιστοσύνης αποτελεσματικά.

Αύξηση της ευκρίνειας του Κρυφού Χώρου Χαρακτηριστικών

Χρησιμοποιούμαι το MaskGST από το πρώτο πείραμα σαν βασικό Μετασχηματιστή (base Transformer). Αυτό το MaskGST προβλέπει οπτικά σύμβολα σε ανάλυση 8×8, χρησιμοποιώντας τον κρυφό χώρο ενός VQ-GAN με $f = 8$. Εκπαιδεύουμε έναν Μετασχηματιστή που προβλέπει οπτικά σύμβολα σε ανάλυση 16×16, υπό συνθήκη Διανυσμάτων Λέξεων (text embeddings), αλλά και των χαμηλότερης ανάλυσης (Low Resolution - LR) οπτικών συμβόλων που προβλέπει το βασικό μοντέλο.

Ο Super-Res Μετασχηματιστής έχει ίδιες υπερπαραμέτρους με το αρχικό μας MaskGST (2 Πλήρη Στρώματα, ακολουθούμενα από 4 Αυτό-Στρώματα, $d = 1024$, $n_{vocab} = 2500$, $n_{heads} = 8$, $lr = 5e - 3$, $n_{epochs} = 200$). Επίσης, όπως έχουμε πει, τα LR οπτικά σύμβολα περνάνε από μιά σειρά Αυτο-Στρωμάτων πριν χρησιμοποιηούν από τον Super-Res Μετασχηματιστή. Για αυτό των σκοπό χρησιμοποιούνται 4 Αυτο-Στρώματα με $d = 1024$ και $n_{heads} = 8$. Ο συνολικός αριμός παραμέτρων, συμπεριλαμβάνοντας το VQ-GAN, αλλά όχι το βασικό μοντέλο είναι 139M.

Στον Πίνακα 9.3 (MaskGST w/ Latent Super Res.) φαίνονται τα αποτελέσματα του πειράματος. Αυτή η μέθοδος δίνει χειρότερα αποτελέσματα σε όλες τις μετρικές σε σχέση με το baseline MaskGST (πρώτο πείραμα). Πιστεύουμε ότι αυτό το εύρημα αποτελεί επιπλέον απόδειξη για τις παρατηρήσεις που κάναμε σχετικά με τα πειράματα στο VQ-GAN στην Ενότητα 1.5.2. Όπως είχαμε παρατηρήσει, η ανάλυση 8×8, στην κρυφή αναπαράσταση είχε με διαφορά τα καλύτερα αποτελέσματα, κάτι που είχαμε αποδόσει στο πόσο λεπτομερή είναι τα χαρακτηριστικά (features) των διανυσμάτων του κρυφού χώρου. Υποθέτουμε, ότι η υψηλότερη ανάλυση (16×16) που χρησιμοποιεί το Super-Res μοντέλο γίνεται υπερβολικά λεπτομερής, κωδικοποιώντας λεπτά χαρακτηριστικά που είναι δύσκολο να αντιστοιχιστούν αποτελεσματικά σε πιο χονδρικές γλωσσικές έννοιες από το Μετασχηματιστή.

Απόπλεξη Χαρακτηριστικών στον Κρυφό Χώρο

Για αυτό το πείραμα εκπαιδεύουμε ένα τροποποιημένο VQ-GAN με δύο βιβλιοθήκες διακριτών διανυσμάτων στον κρυφό χώρο, $e^{char}, e^{background} \in \mathbb{R}^{128 \times 256}$ όπως περιγράψαμε στην Ενότητα 1.4.7. Το VQ-GAN κωδικοποιεί μία εικόνα με βάση την βιβλιοθήκη Χαρακτήρων και Φόντου και συνδυάζει τις κωδικοποιήσεις χρησιμοποιώντας την Μάσκα Φόντου.

Συνδυάζουμε το VQ-GAN με τις δύο βιβλιοθήκες με έναν Μετασχηματιστή, όπως περιγράφεται στην Ενότητα 1.4.7. Οι υπερπαράμετροι του Μετασχηματιστή μένουν ίδιες με του αρχικού MaskGST. Η διαφορά είναι ότι έχουν τρεις γραμμικούς μετασχηματισμού στην έξοδο, για να αντιστοιχίζουν την έξοδο του Μετασχηματιστή σε logits Χαρακτήρων, logits Φόντου και Μάσκα Φόντου.

Τα αποτελέσματα για αυτό το πείραμα (Πίνακας 9.3 - MaskGST w/ latent space disentanglement) είναι απογοητευτικά σε όλες τις μετρικές. Το FID ανεβαίνει, ενώ οι υπόλοιπες μετρικές ελλατώνονται, σε σύγκριση με το πρωτότυπο MaskGST (baseline). Υποθέτουμε ότι αυτό έχει να κάνει με την πολυπλοκότητα της μοντελοποίησης για τον Μετασχηματιστή. Συγκεκριμένα, αυτός καλείται να παραμετροποιήσει τόσο την $p_\theta(z_{char}|T)$, όσο και την $p_\theta(z_{background}|T)$ (το θ αναπαριστά τις παραμέτρους τους Μετασχηματιστή και το T την αναπαράσταση των κειμενικών περιγραφών). Δεδομένου ότι τα z_{char} και $z_{background}$ μοντελοποιούνται μέσω διαφορετικών χώρων αναπαράστασης (διαφορετικές βιβλιοθήκες διανυσμάτων) είναι, ίσως ανέφικτο για έναν Μετασχηματιστή να μάθει και τις δύο κατανομές ταυτόχρονα.

Συνδυασμοί Μεθόδων

Μέσω των προηγούμενων πειραμάτων εντοπίσαμε κάποιες υποσχόμενες ιδέες. Σε αυτή την ενότητα, αξιολογούμε αν οι μέθοδοι που δούλεψαν καλά ανεξάρτητα μπορούν να συνδυαστούν για να ενισχύσουν η μία την άλλη και να φέρουν περαιτέρω βελτίωση. Συγκεκριμένα υπάρχουν τρεις μέθοδοι που βελτίωσαν τα αποτελέσματα σε σχέση με το baseline MaskGST:

- MaskGST-SV (config b)
- MaskGST με επαύξηση γλωσσικών δεδομένων
- MaskGST-CG$_{\pm}$ (Καθοδήγηση Χαρακτήρων και χρήση Αρνητικών Υποδείξεων)

Συνδυάζουμε τις παραπάνω μεθοδους σε ζευγάρια και όλες μαζί. Στον Πίνακα 9.2 συγκεντρώνουμε τα αποτελέσματα για αυτά τα πειράματα. Συμπεριλαμβάνουμε και τα αποτελέσματα των επιμέρους πειραμάτων για ευκολία σύγκρισης. Προκύπτει ότι ο καλύτερος συνδιασμός είναι: *MaskGST-CG-$_{\pm}$ w/ aug. captions*. Συγκεκριμένα, βελτιώνει το FID για περισσότερες από 3 μονάδες και το Char-F1 για σχεδόν 1 μονάδα, σε σύγκριση με το *MaskGST-CG$_{\pm}$*. Το Char-Acc μένει ιδιο. Αξίζει να σημειωθεί ότι το BLEU-score είναι ελαφρώς χειρότερο από του *MaskGST-CG$_{\pm}$*.

1.5.3 Πειράματα Υπερ-Παραμέτρων

Υπερ-Παράμετροι του Μετασχηματιστή

Έχοντας φτάσει στην καλύτερη αρχιτεκτονική μας: *MaskGST-CG-$_{\pm}$ w/ aug. captions*, πραγματοποιούμε κάποια μη εξαντλητικά πειράματα όσον αφορά τις υπερπαραμέτρους. Συγκεκριμένα, εξερευνούμε την επίδραση αλλαγών στο **μήκος** (πλήθος στρωμάτων) και το **πλάτος** (κρυφή διάσταση) του Μετασχηματιστή, στα αποτελέσματα.

Ο Πίνακας 9.4 δείχνει τα αποτελέσματα των πειραμάτων, όσον αφορά το μήκος του Μετασχηματιστή. Κρατάμε τις υπόλοιπες υπερ-παραμέτρους σταθερές (η διάσταση είναι $d = 1024$) και κλιμακώνουμε το μήκος από τα 4 μέχρι τα 16 στρώματα. Παρατηρούμε ότι η άυξηση των στρωμάτων οδηγεί σε βελτίωση όλων των μετρικών.

Στον Πίνακα 9.5, συγκεντρώνουμε τα αποτελέσματα , όσων αφορά τα πειράματα για την διάσταση του Μετασχηματιστή. Για αυτά τα πειράματα κρατάμε όλες τις άλλες υπερ-παραμέτρους ίδιες και ποικίλουμε την κρυφή διάσταση του μοντέλου από $d = 768$ to $d = 2048$. Ο αριθμός των στρωμάτων είναι ίσος με 6 (2 Πλήρη Στρώματα ακολουθούμενα από 4 Αυτο-Στρώματα). Η αύξηση της διάστασης του μοντέλου συνδέεται με βελτίωση σε όλες τις μετρικές. Συγκεκριμένα, ο Μετασχηματιστής με $d = 2048$ είναι το καλύτερο μοντέλο μας.

Μελέτη για την Καθοδήγηση Χαρακτήρων

Χρησιμοποιώντας το καλύτερο μοντέλο μας, *MaskGST-CG$_{\pm}$ w/ aug.captions* (d=2048), πραγματοποιούμε μία μελέτη πάνω στον παράγοντα (f) της Καθοδήγησης Χαρακτήρων (Εξίσωση 8.3.8). Τα αποτελέσματα συνοψίζοται στην Εικόνα 1.5.1. Οι καμπύλες που αντιπροσωπεύουν τις Μετρικές που σχετίζονται με του Χαρακτήρες έχουν το ίδιο ακριβώς σχήμα. Είναι αύξουσες μέχρι το $f = 0.6$, ενώ μειώνονται ελαφρώς για το $f = 0.8$. Το FID, από την άλλη, μειώνεται (βελτιώνεται) μέχρι το $f = 0.4$ και στην συνέχεια αυξάνεται για τις δύο επόμενες περιπτώσεις. Σε διαισθητικό επίπεδο, βλέπουμε ότι με την αύξηση του f βελτιώνουμε την παραγωγή των Χαρακτήρων. Αυτό είναι αναμενόμενο, αφου με την άυξηση αυτή, το μοντέλο δίνει όλο και περισσότερη προσοχή στα logits των Χαρακτήρων σε σχέση με τα logits που βασίζονται στην κειμενική περιγραφή. Βελτιώνοντας τους Χαρακτήρες, βελτιώνουμε και την συνολικη ποιότητα των εικόνων, μέχρι κάποιον βαθμό, αφού αυτοί αποτελούν βασικό κομμάτι των περισσότερων εικόνων. Αυτό εξηγεί το γεγονός ότι η άυξηση του f μέχρι κάποιο σημείο ($f = 0.4$) βελτιώνει και το FID (ποιότητα των εικόνων). Ωστόσο, η περαιτέρω αύξηση του f, έχει αρνητικό αντίκτυπο στην συνολική ποιότητα των εικόνων, καθώς δύνουμε δυσανάλογα πολύ βάρος στα logits των Χαρακτήρων και παραμελούμε τα logits που βασίζονται στις περιγραφές. Τότε το μοντέλο αποτυγχάνει να κατασκευάσει κομμάτια άλλα κομμάτια στην εικόνα, που δεν σχετίζονται με τους Χαρακτήρες.

Για $f = 0.4$ παίρνουμε τον βέλτιστο συνδυασμό μετρικών (68.32, 41.40 και 42.49 για τα Char-F1, Char-Acc and FID, αντίστοιχα). Ωστόσο, στην πράξη, βρίσκουμε ότι ακόμα και το $f = 0.4$ κάνει το μοντέλο να αφιερώσει υπερβολικό προσοχή στην παραγωγή των Χαρακτήρων, με αρνητικά αποτελέσματα στην συνολική ποιότητα την ιστορίας, λόγω της απώλεια της συνοχής των εικόνων. Κρίνουμε ότι με $f = 0.2$ επιτυγχάνεται ο καλύτερος

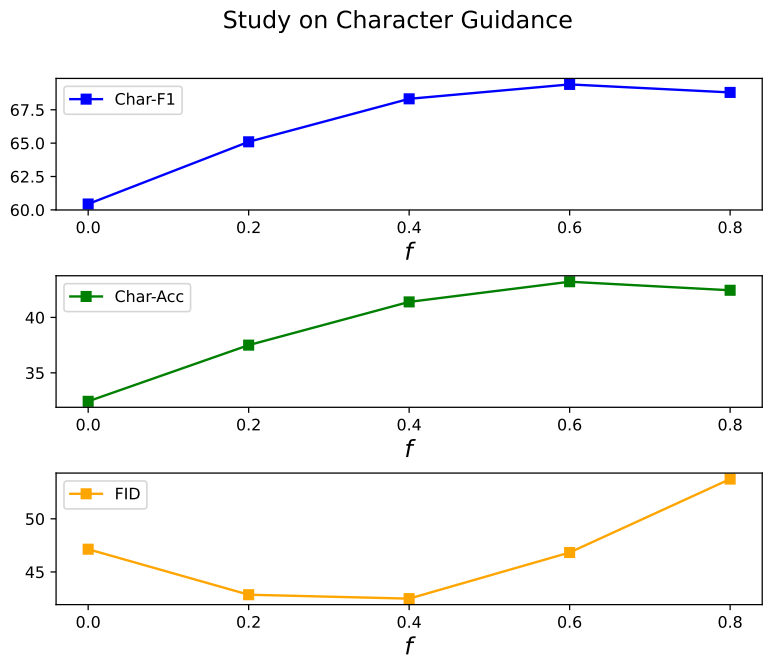


Figure 1.5.1: Μελέτη του παράγοντα f στην Καθοδήγηση Χαρακτήρων.

συμβιβασμός (trade-off) μεταξύ παραγωγής των Χαρακτήρων και συνολικών ποιότητα της Ιστορίας, σε ποιοτικό επίπεδο.

1.5.4 Σύγκριση με Προηγούμενες Τεχνικές

Σε αυτό το σημείο συγκρίνουμε τα μοντέλα μας με προηγούμενες τεχνικές. Όσον αφορά τα GANs, συγκρίνουμε με τα StoryGAN [18], CP-CSV[36], DUCO-StoryGAN[20] και VLC-StoryGAN[19]. Επίσης συγκρίνουμε με τις δύο αρχιτεκτονικές τύπου Μετασχηματισή: VP-CSV [5] και CMOTA[1]. Αναφορικά με τα μοντέλα διάχυσης, υπάρχουν τρεις προηγούμενες τεχνικές: AR-LDM[22], ACM-VSG[10] και Causal-Story[35]. Η άμεση σύγκριση με αυτά τα μοντέλα δεν είναι δίκαιη, δεδομένου ότι βασίζονται στο LDM[28], που είναι προ-εκπαιδευμένο σε ένα τεράστιο σύνολο δεδομένων με άφθονες υπολογιστικές υποδομές. Επιπλέον, απαιτούν πολύ ακριβό εξοπλισμό για την ανάπτυξή τους. Ενδεικτικά, το AR-LDM χρησιμοποιεί [40×] την vRAM που χρησιμοποιείται στην δική μας δουλειά. Ωστόσο, για λόγους πληρότητας, συμπεριλαμβάνουμε αυτά τα μοντέλα στην σύγκριση που κάνουμε. Αναφερόμαστε σε αυτά ξεχωριστά στην υποενότητα 1.5.4.

Ο πίνακας 9.6 συγκεντρώνει τα αποτελέσματα των προηγούμενων μεθόδων μαζί με το βασικό μας μοντέλο (baseline) MaskGST, καθώς και το καλύτερο μοντέλο μας MaskGST-CG$_{\pm}$, with $d = 1024$ and $d = 2048$.

Όπως έχουμε ήδη αναφέρει, το StoryLDM και το StoryGPT-V εφαρμόζονται σε μία παραλαγμένη έκδοχή της SV, όπου επανειλημμένες αναφορές στους χαρακτήρες, αντικαθιστώνται με αντωνυμίες. Για αυτόν τον λόγο, η άμεση σύγκριση μαζί τους δεν είναι εφαρμόσιμη.

MaskGST-CG$_{\pm}$ w/ aug. captions ($d = 1024$) Το μοντέλο μας με $d = 1024$ αποδίδει καλύτερα από όλες τις προηγούμενες προσεγγίσεις με GANs η Μετασχηματιστές, σε όλες τις μετρικές (χαμηλότερο FID και υψηλότερα Char-F1, Char-Acc, BLEU-2/3). Ειδικά στις μετρικές που αφορούν τους χαρακτήρες, η βελτίωση είναι σημαντική. Συγκεκριμένα, το Char-F1 αυξάνεται για 3.6 και το Char-Acc για 7.7 μονάδες, σε σχέση με το προηγούμενο καλύτερο (VP-CSV). Θεωρούμε ότι η υπεροχή του μοντέλου μας σε αυτές τις μετρικές μπορεί να αποδοθεί, σε μεγάλο βαθμό στην τεχνική Καθοδήγησης Χαρακτήρων που προτείνουμε.

MaskGST-CG$_{\pm}$ w/ aug. captions ($d = 2048$) Η διπλασιασμός της κρυφής διάστασης σε $d = 2048$ βελτιώνει τα αποτελέσματά μας σε όλες τις μετρικές. Ιδιαίτερα σημαντική είναι η μείωση του FID, το οποίο μειώνεται κατά 8.8 μονάδες συγκριτικά με την έκδοση του μοντέλου με $d = 1024$. Υποθέτουμε ότι ο διπλασιασ-

μός της κρυφής διάστασης αφήνει περισσότερο χώρο στο μοντέλο για να μάθει πιο πολύπλοκες και λεπτομερείς αντιστοιχίσεις μεταξύ λέξεων και οπτικών χαρακτηριστικών, που έχουν ως αποτέλεσμα εικόνες υψηλότερης ποιότητας με περισσότερες λεπτομέρειες. Η ικανότητα του μοντέλου να μάθει πιο πολύπλοκες αναπαραστάσεις και να παράγει πιο λεπτομερείς εικόνες μπορεί να εξηγήσει τις βελτιώσεις και στις άλλες μετρικές. Ειδικότερα, οι εικόνες υψηλής ποιότητας θα απεικονίζουν βελτιωμένες εκδοχές των χαρακτήρων και θα παράγουν καλύτερα σκορ BLEU, μέσω του υποτιτλισμού των παραγόμενων εικόνων.

Σύγκριση με Μοντέλα Διάχυσης Όπως γίνεται φανερό από τον Πίνακα 9.6, υϊοθετούν μόνο το FID σαν μετρική. Αυτό δεν είναι επαρκές για την Οπτικοποίηση Ιστορίας, δεδομένου ότι λαμβάνει υπόψιν μόνο των ποιότητα κάθε επιμέρους εικόνας. Όσων αφορά το FID, τα τρία μοντέλα διάχυσης είναι ασυναγώνιστα σε σύγκριση με όλες τις υπόλοιπες προσεγγίσεις. Ωστόσο αυτό μπορεί να αποδοθεί-τουλάσιστον σε έναν βαθμό-στην εκτεταμένη προεκπαίδευσή τους, καθώς και στα μεγάλα πλήθη παραμέτρων.

1.5.5 Ποιοτικά αποτελέσματα

PororoSV

Στην Εικόνα 1.5.2 παρέχουμε τέσσερα παραδείγματα από ιστορίες εικόνων, από το σύνολο επαλήθευσης του Pororo-SV. Για κάθε ιστορία παρουσιάζουμε τις λεζάντες, τις πρωτότυπες εικόνες, τις εικόνες που παράγονται από το CMOTA[1] και τις εικόνες από το μοντέλο μας (MaskGST-CG$_{\pm}$ w/ aug. captions ($d = 2048$)). Για αυτόν τον σκοπό, χρησιμοποιούμαι το προ-εκπαιδευμένο CMOTA που έχει δημοσιευθεί εδώ[4]. Πραγματοποιούμαι μία ποιοτική σύγκριση με βάση αυτά τα ποιοτικά αποτελέσματα.

Οπτική Ποιότητα Όσον αφορά την οπτική ποιότητα, είναι εμφανές από τα παραδείγματα ότι το μοντέλο μας είναι ανώτερο από το CMOTA. Για παράδειγμα, στο πάνω-αριστερό πλαίσιο, η πρώτη και η τελευταία εικόνα, που κατασκευάστηκαν από το CMOTA είναι θολές και ανεπαρκώς κατανοητές. Αντίθετα, στη δική μας περίπτωση, και οι δύο εικόνες περιέχουν αναγνωρίσιμα αντικείμενα (χαρακτήρες). Όσον αφορά τις τρεις ενδιάμεσες εικόνες, αυτές που παράγει το CMOTA περιέχουν μερικούς αναγνωρίσιμους χαρακτήρες. Ωστόσο, ακόμα και σε αυτή την περίπτωση, οι εικόνες μας είναι πολύ υψηλότερης ποιότητας, με τους χαρακτήρες να έχουν σημαντικά πιο λεπτομερείς εμφανίσεις (π.χ. τα μάτια του Crong (του πράσινου δεινοσαύρου) και το ράμφος του Pororo (του πιγκουίνου)).

Χρονική Συνέπεια Όσον αφορά στην χρονική συνέπεια, παρατηρούμε ότι οι εικόνες του CMOTA δυσκολεύονται να διατηρήσουν σταθερό φόντο και στα 4 παραδείγματα. Ειδικότερα, εικόνες από εξωτερικό χώρο εναλλάσονται με εικόνες από εσωτερικό χώρο. Για παράδειγμα, στο πάνω δεξί πάνελ, η πρώτη και η τελευτάια εικόνα φαίνεται να δείχνουν ένα χιονισμένο φόντο, ενώ οι άλλες τρεις φαίνεται να είναι από εσωτερικό χώρο. Αντίθετα, το μοντέλο μας καταφέρνει να διατηρήσει έναν σχετικά συνεκτικό φόντο στις περισσότερες περιπτώσεις. Ειδικά στο πάνω δεξί πάνελ, η εμφάνιση του δωματίου διατηρείται εξαιρετικά συνεπής. Να σημειώσουμε, ότι στο κάτω δεξί πάνελ, το μοντέλο μας αντιμετωπίζει δυσκολίες στη συνέπεια και ειδικά στην τέταρτη εικόνα που έχει άσχετο φόντο, σε σύγκριση με τις γειτονικές.

Σημασιολογική Συνάφεια Ο όρος *Σημασιολογική Συνάφεια* αναφέρεται στο αν οι παραγόμενες εικόνες είναι σχετικές με τις αντίστοιχες λεζάντες. Όσον αφορά αυτό, το CMOTA φαίνεται να αντιμετωπίζει ιδιαίτερα προβλήματα στις περιπτώσεις όπου αναφέρονται πολλοί χαρακτήρες. Για παράδειγμα, στο κάτω-δεξιό πάνελ, η πρώτη λεζάντα αναφέρει τους Petty και Loopy, αλλά δημιουργείται μόνο ο Petty. Στη δεύτερη λεζάντα, όπου αναφέρονται πολλοί χαρακτήρες, η εικόνα του CMOTA είναι ακατανόητη. Στις επόμενες τρεις εικόνες, το CMOTA καταφέρνει να δημιουργήσει τους περισσότερους αναφερόμενους χαρακτήρες, αν και με χαμηλή ποιότητα. Αντίθετα, το μοντέλο μας καταφέρνει να δημιουργήσει όλους τους σχετικούς χαρακτήρες στις περισσότερες περιπτώσεις. Στο κάτω-δεξιό πάνελ, αυτό ισχύει για όλες τις λεζάντες. Είναι επίσης αξιοσημείωτο ότι, στη δεύτερη εικόνα, καταφέρνει να δημιουργήσει πολλούς χαρακτήρες με εξαιρετική ποιότητα, καθώς και το κόκκινο αυτοκίνητο που αναφέρεται στη λεζάντα

[4] https://github.com/yonseivnl/cmota

Frame 1: Crong eats meat. Loopy gave a dish of vegetables.
Frame 2: Crong doesn't like vegetables. Crong pushed the dish of vegetables.
Frame 3: Both Pororo and Crong look at the meat in the dish.
Frame 4: Crong picked up the last meat.
Frame 5: Pororo told that Crong is so greedy.

Frame 1: Loopy is back with Poby.
Frame 2: Loopy and Poby notice that chair is not broken.
Frame 3: Poby and Loopy thinks the chair is very strange.
Frame 4: Loopy is looking at chair with question.
Frame 5: Loopy thanks Poby for coming.

Frame 1: Petty asks if Crong really okay is.
Frame 2: Loopy says Crong looks not that good.
Frame 3: Crong trying to pretend to be okay waves Crong head.
Frame 4: Crong pretends to do some freehand exercise.
Frame 5: Pororo laughs at Crong calling Crong as a regular pooping machine.

Frame 1: Loopy talks and spreads Petty arms. Petty looks at Loopy smiles and nods.
Frame 2: Poby gathers red car hands and talks. Loopy Petty and Harry are looking at Poby.
Frame 3: Loopy Poby and Petty walk stop and turn back.
Frame 4: Harry looks angry and talks.
Frame 5: Harry looks at Poby and turns red car head.

Figure 1.5.2: Ποιοτική σύγκριση μεταξύ του μοντέλου μας(MaskGST-CG$_{\pm}$ w/ aug. captions) και του CMOTA[1] σε τέσσερα παραδείγματα ιστοριών.

1.5.6 Ανθρώπινη Αξιολόγηση

Για την περαιτέρω διερεύνηση των ποιοτικών αποτελεσμάτων μας, διεξάγαμε μια έρευνα, βασισμένη σε ανθρώπινη αξιολόγηση, με βάση τρία κριτήρια που έχουν υϊοθετηθεί και σε προηγούμενες δουλειές[20, 19, 1], συγκρίνοντας το μοντέλο μας με το CMOTA [1]. Τα τρία κριτήρια είναι τα εξής:

- Η **Οπτική Ποιότητα**, που αναφέρεται στο κατά πόσο οι εικόνες είναι οπτικά ευχάριστες, σε αντίθεση με το να είναι θολές και δυσνόητες.

- Η **Χρονική Συνέπεια** σχετίζεται με το αν οι εικόνες είναι συνεπείς μεταξύ τους, κρατώντας ένα κοινό θέμα και σχηματίζοντας μία ιστορία, αντί να μοιάζουν σαν πέντε ξεχωριστές σκηνές.

- Η **Σημασιολογική Συνάφεια** αναφέρεται στο κατά πόσο οι εικόνες αντικατοπτρίζουν τις αντίστοιχες λεζάντες και τους χαρακτήρες που αναφέρονται σε αυτές, με ακρίβεια.

Η αξιολόγηση γίνεται με βάση 100 ιστορίες από το σύνολο επαλήθευσης του Pororo-SV. Κάθε ιστορία αξιολογείται από δύο διακριτούς χρήστες. Τα αποτελέσματα της έρευνας (Table 1.1) δείχνουν ότι το μοντέλο μας υπερέχει με βάση όλα τα κριτήρια, κάτι που υποστηρίζει και τα ποσοτικά μας αποτελέσματα.

Κριτήριο	Ours (%)	CMOTA (%)	Tie(%)
Οπτική Ποιότητα	**78%**	3%	19%
Χρονική Συνέπεια	**66%**	8%	26%
Σημασιολογική Συνάφεια	**64%**	9%	27%

Table 1.1: Τα αποτελέσματα της ανθρώπινης αξιολόγησης. Συγκρίνουμε το μοντέλο μας *MaskGST-CG$_{\pm}$ w/ aug. caption (d=2048)* (Ours) με το CMOTA, με βάση τρία κριτήρια. το Our(%) και το CMOTA(%) αντιστοιχούν στα ποσοστά των περιπτώσεων που το κάθε μοντέλο επιλέγεται και από τους δύο χρήστες, ενώ το Tie(%) αντιστοιχεί στις υπόλοιπες περιπτώσεις.

1.5.7 Ανάλυση Χρησιμοποιούμενων Πόρων

Πόροι κατά την Εκπαίδευση

Χρησιμοποιούμε περίπου 36 και 107 ώρες αντίστοιχα για να εκπαιδεύσουμε τους Μετασχηματιστές με $d = 1024$ and $d = 2048$, αντίστοιχα σε μία NVIDIA V100 (16GB). Αυτό αντιστοιχεί σε (36 hours) $\cdot$ (16GB) = 576 (GB $\cdot$ hours) και (107 hours)$\cdot$ (16GB) = 1712 (GB $\cdot$ hours) χρήσης GPU, αντίστοιχα. Για σύγκριση, για το VP-CSV [5] αναφέρεται ότι χρησιμοποούνται 4 NVIDIA A100 (40GB) για 12 ώρες. Αυτό αντιστοιχεί σε (12 hours) $\cdot$ (4$\cdot$40 GB) = 1920 (GB $\cdot$ hours) χρήσης GPU, χωρίς να λάβουμε υπόψιν ότι η A100 είναι πιο σύγχρωνη GPU από την V100. Το CMOTA [1] δεν αναφέρει χρήση πόρων κατά την εκπαίδευση.

Πόροι κατά τον Συμπερασμό

Δεδομένου ότι πραγματοποιήσαμε συμπερασμό τόσο για τα μοντέλα μας, όσο και για το CMOTA, στην ίδια GPU, μπορούμε να κάνουμε μία δίκαιη σύγκριση. Τα μοντέλα μας με $d = 1024$ και $d = 2048$ χρειάζονται 34 λεπτά και 94 λεπτά, αντίστοιχα για να πραγματοποιήσουν συμπερασμό για τις 2208 ιστορίες του συνόλου επαλήθευσης. Αυτό αντιστοιχεί σε 0.92 *sec/story* και 2.55 *sec/story*. Για την ίδια εργασία, το CMOTA ξοδεύει 228 λεπτά, που αντιστοιχεί σε 6.19 *sec/story*. Είναι φανερό ότι τα μοντέλα μας είναι σημαντικά πιο αποδοτικά από το CMOTA. Αυτό μπορεί, σε μεγάλο βαθμό να αποδοθεί στο σχήμα συμπερασμού των Μετασχηματιστών, τύπου MaskGIT, που παράγουν πολλά οπτικά σύμβολα ανά βήμα, σε αντίθεση με τους αυτο-τροφοδοτικούς Μετασχηματιστές, όπως το CMOTA, που προβλέπουν τα οπτικά σύμβολα, ένα-ένα.

1.6 Συμπεράσματα και Μελλοντικές Κατευθύνσεις

1.6.1 Συμπεράσματα

Σε αυτή την εργασία διερευνήσαμε την χρήση Μετασχηματιστών τύπου MaskGIT, για την Οπτικοποίηση Ιστορίας, για πρώτη φορά. Τα αποτελέσματα μας, που είναι τα καλύτερα που έχουν επιτευχθεί, σε διάφορες μετρικές αποτελούν απόδειξη για αξία της προσέγγισής μας, όσον αφορά την SV. Άλλωστε, αναδεικνύουν ότι τέτοιες αρχιτεκτονικές αξίζει να διερευνηθούν περαιτέρω στο πλαίσιο παραγωγικών εργασιών υπολογιστικής όρασης, γενικότερα.

Συγκεκριμένα, δημιουργήσαμε το βασικό μας μοντέλο MaskGST, βασισμένο στο MaskGIT, με επιπλέον μηχανισμούς Ετέρο-Προσοχής, προκειμένου οι δημιουργηθείσες εικόνες σε κάθε στάδιο της ιστορίας να επηρεάζονται από παλαιότερες και μελλοντικές εικόνες. Πειραματιστήκαμε με διάφορες τροποποιήσεις της αρχικής αρχιτεκτονικής.

Αρκετά από αυτά τα πειράματα απέτυχαν να προσφέρουν βελτιώσεις σε σύγκριση με το βασικό μοντέλο. Η χρήση του T5-XXL ως κωδικοποιητής κειμένου πιθανόν δεν είναι βέλτιστη λόγω των εξαιρετικά εξειδικευμένων περιγραφών κειμένου του συνόλου δεδομένων μας, των οποίων οι ονομασίες χαρακτήρων είναι ιδιαίτερα ασυνήθιστες. Υποψιαζόμαστε ότι η προσπάθειά μας να εκτελέσουμε αύξηση ευρκίνειας στον Κρυφό Χώρο Χαρακτηριστικών απέτυχε, διότι τα διακριτικά εικονοσύμβολα υψηλής ανάλυσης είναι πολύ λεπτομερή για τον Μετασχηματιστή να προβλέψει βάσει απλών προτροπών κειμένου. Όσον αφορά τον Κριτή Συμβόλων Βασισμένο σε Χαρακτήρες που σχεδιάσαμε, πιστεύουμε ότι απέτυχε επειδή η συνθήκη μόνο σε διανύσματα Χαρακτήρων δεν είναι αρκετά εκφραστική. Τέλος, υποψιαζόμαστε ότι η προσπάθειά μας να πραγματοποιήσουμε την απόπλεξη του χώρου των χαρακτηριστικών των χαρακτήρων από τα χαρακτηριστικά του φόντου απέτυχε, επειδή κατέληξε να είναι πολύ πολύπλοκο για τον Μετασχηματιστή να το μοντελοποιήσει.

Από την άλλη πλευρά, μερικά από τα πειράματά μας έδωσαν ελπιδοφόρα αποτελέσματα. Η ενσωμάτωση SV-στρωμάτων στον Μετασχηματιστή ωφελήθησε όλες τις μετρικές, με το να αντιμετωπίζει όλα τα εικονοσύμβολα και τα κείμενα μιας ιστορίας ως μια συνεχή ακολουθία, για μια μερίδα της δημιουργικής διαδικασίας. Δεύτερον, προτείναμε μια απλή τεχνική για την επαύξηση κειμενικών δεδομένω, ανεξάρτητα από τις εικόνες, χρησιμοποιώντας ένα LLM. Αυτή η ιδέα βελτίωσε τα αποτελέσματά μας μειώνοντας τον κίνδυνο υπερ-προσαρμογής και βοηθώντας το μοντέλο να επικεντρωθεί σε σημαντικές γλωσσικές έννοιες. Τέλος, η μέθοδος Καθοδήγησης Χαρακτήρων που σχεδιάσαμε επέδειξε τα πιο ελπιδοφόρα αποτελέσματα. Με το να δημιουργεί τρια ξεχωριστά συνόλα logits, ένα βάσει συνθηκών κειμένου, ένα με βάση το θετικό σύνολο Χαρακτήρων και ένα με βάση το αρνητικό σύνολο Χαρακτήρων και στη συνέχεια να τα συνδυάζει, καταφέρνει να καθοδηγεί το μοντέλο προς την υψηλής ποιότητας δημιουργία χαρακτήρων, διατηρώντας ταυτόχρονα και άλλες πληροφορίες από τις λεζάντες.

Συνδυάζοντας ελπιδοφόρες μεθόδους και προσαρμογή υπερπαραμέτρων, φτάσαμε στο καλύτερο μας μοντέλο, *MaskGST-CG$_{\pm}$ w/ aug. captions (d=2048)*. Αυτό το μοντέλο, υπερβαίνει το προηγούμενο μοντέλο Μετασχηματιστή κατά 9,3, 11,8 και 12,8 μονάδες σε σχέση με τα FID, Char-F1 και Char-Acc αντίστοιχα. Παρά το γεγονός ότι είναι μεγαλύτερο από τους προηγούμενους Μετασχηματιστές, είναι πιο αποδοτικό χρονικά τόσο στην εκπαίδευση όσο και στον συμπερασμό.

1.6.2 Μελλοντικές Κατευθύνσεις

Πιστεύουμε ότι τα αποτελέσματά μας υποδεικνύουν ισχυρά την ανταγωνιστικότητα των Μετασχηματιστών τύπου MaskGIT για την εργασία της Οπτικοποίησης Ιστοριών. Το έργο μας ανοίγει το δρόμο για περαιτέρω πειραματισμούς με τέτοιες αρχιτεκτονικές, είτε με την αύξηση του μεγέθους των μοντέλων μας, είτε με την εξερεύνηση άλλων δυνατών τροποποιήσεων. Επιπλέον, οι αρχιτεκτονικές του MaskGIT έχουν εξεταστεί σχετικά λίγο ακόμα και στον χώρο της δημιουργίας Εικόνας από Κείμενο (Text-to-Image). Ελπίζουμε ότι το έργο μας μπορεί να ενθαρρύνει την εξερεύνησή τους και σε αυτό τον τομέα.

Επιπλέον, η μέθοδος μας για την επάυξηση γλωσσικών περιγραφών παρέχει ένα εύκολο στη χρήση πλαίσιο για την εμπλουτισμό συνόλων δεδομένων που περιλαμβάνουν κείμενο, συμπεριλαμβανομένης, αλλά όχι περιοριστικά, της Οπτικοποίησης Ιστοριών και της Δημιουργίας Κειμένου προς Εικόνα. Επιπλέον, καθώς τα Μεγάλα Γλωσσικά Μοντέλα γίνονται περισσότερο διαθέσιμα, η μέθοδός μας θα μπορούσε να επεκταθεί με την εξαγωγή εναλλακτικών λεζάντων, χρησιμοποιώντας πολλαπλά LLMs, για να επιτύχει μεγαλύτερη ετερογένεια ως προς τις περιγραφές κειμένου.

Όσον αφορά τη μέθοδο Καθοδήγησης Χαρακτήρων, θεωρούμε ότι αξίζει περαιτέρω έρευνα. Από τη μία πλευρά, μπορεί να δοκιμαστεί σε διαφορετικά πλαίσια, για την εργασία της Οπτικοποίησης Ιστοριών. Ένας πιθανός τρόπος να γίνει αυτό είναι να ενσωματωθεί σε (μεγάλα) προ-εκπαιδευμένα μοντέλα, είτε βασισμένα σε διάχυση είτε σε Μετασχηματιστές. Αυτό θα περιλαμβάνει την προσθήκη θετικού και αρνητικού διανύσματος για κάθε χαρακτήρα στο μοντέλο και την εκπαίδευσή του με το σχήμα εκπαίδευσής μας, όπου οι συνθήκες κειμένου απορρίπτονται για μια μερίδα των δειγμάτων εκπαίδευσης. Δεδομένου ότι ο αριθμός των επιπλέον παραμέτρων είναι μικρός, αυτό θα μπορούσε να λειτουργήσει με σχετικά λίγη επιπλέον εκπαίδευση (fine-tuning). Υποθέτουμε ότι ο συνδυασμός καλύτερης κατανόησης των γλωσσικών περιγραφών (prompts), λόγω μακράς προεκπαίδευσης, με την αποτελεσματική μας μέθοδο καθοδήγησης χαρακτήρων θα μπορούσε να παράγει εντυπωσιακά αποτελέσματα για την εργασία.

Τέλος, μια άλλη πιθανή ερευνητική πορεία θα μπορούσε να είναι η γενίκευση της μεθόδου Καθοδήγησης Χαρακτήρων σε άλλες εργασίες. Ειδικότερα, οποιαδήποτε γεννητική εργασία όπου ενδιαφερόμαστε ρητά για την δημιουργία ενός συγκεκριμένου συνόλου έννοιων (Χαρακτήρες στην περίπτωση της SV) θα μπορούσε ενδεχομένως να επωφεληθεί από την υιοθέτηση μιας τέτοιας μεθόδου.

Chapter 2

Introduction

Over the past decade, the advancement in hardware and software engineering, the accumulation of vast amounts of data as well as the rigorous research in deep learning have snowballed into a unprecedented growth and adoption of AI systems. Especially in 2023, it has been made clear that this phenomenon is justly considered to be the next great technological revolution. The release of - the now famous - ChatGPT in late 2022 and its blistering rise to fame, brought AI from the scientific domain out into the public spotlight. It became part of our everyday lives, a pop-culture phenomenon and a subject of political discourse. ChatGPT, among other LLMs (Large Language Models) (e.g. [13, 37]) has shown exceptional results in terms of language understanding and generation, sometimes even being indistinguishable from humans in such tasks.

2.1 Text-to-Image generation

Apart from language, images are one of the most prominent areas of human experience that serve as a means of communication and expression. In this context, image understanding, as well as generation are equally important challenges for Artificial Intelligence to tackle, in order to come closer to what constitutes *Intelligence*, for human beings. In terms of Image Generation, there has been notable progress in the past years starting with GANs that where introduced in 2014 [11] and dominated the image-generation scene for several years [45, 46]. More recently, transformers [26, 3, 4], as well as diffusion models [27, 30, 28] have revolutionalized image generation, by improving the visual quality and the range of visual themes that can be generated.

2.2 Story Visualization

In this thesis, we focus on the task of Story Visualization (SV). The task consists of generating a sequence of images, each one of which corresponds to a sentence in a given sequence of sentences. The sentences form a coherent textual narrative. It was introduced in 2019 by the authors of [18], who also proposed StoryGAN, the first model to tackle the task.

In a sense, SV can be seen as an extension of Text-to-Image generation, by addition of a temporal aspect in the task. Alternatively, one could think of it as a stepping stone that will eventually lead to long-range Text-to-Video generation, since SV is concerned with small sequences of images (typically 4-5), in contrast to movies that include thousands of frames.

There are two main challenges in SV. Firstly, the sequence of generated images must include the objects and actions referenced in the corresponding sentences and depict them with high quality. Secondly, there is the story aspect of the task; objects that appear in more than one picture must hold a consistent appearance, so that they can be recognised as the same object across the sequence. The most prominent such objects are the actors (main characters) in the stories.

2.3 Contribution

Our main contribution is the adoption of a MaskGIT [3] model for the task of SV, which we enhance with Cross-Attention sub-layers. We are the first to use this architecture for this specific task, which is arguably under-explored compared to Diffusers, even in the task of Text-to-Image generation.

Using this modified MaskGIT as our core model, we also experiment with modifications on various components of the initial architecture, in search of reasearch directions that can improve the quality of the generated stories. Through our experiments, we arrive at our top-performing architecture, *MaskGST-CG$_{\pm}$ w/ aug captions*, that adopts an LLM-driven caption augmentation technique and a Character Guidance mechanism, that we propose for the first time. This model achieves SOTA results over several metrics, on the most prominent SV Dataset, Pororo-SV.

The outline of this thesis is as follows:

- In Chapter 3 we discuss previous works for the task of Story Visualization, from the introduction of the task until recently.
- In Chapter 4 we describe the original Transformer framework, since the Transformer is a core component in our approach.
- Subsequently, Chapter 5 is concerned with the original VQ-VAE architecture. The VQ-VAE is the second main component in our approach.
- Chapter 6 is dedicated to some notable works in Text-to-Image generation that pair Transformers with VQ-VAEs, including [3] and [4], from which we are greatly influenced.
- Chapter 7 provides a brief overview of a caption augmentation technique that is closely related to the one we use.
- In Chapter 8 we provide a detailed review of our approach for the task of Story Visuazation.
- In Chapter 9 we present the experiments that we conducted and comment on the results.
- Finally, in Chapter 10 we review our results as a whole and discuss possible future directions, based on them.

Chapter 3

Previous Work on Story Visualization

In this Chapter we discuss Previous Works on Story Visualization. Since the introduction of the task in 2019, several papers have been published, proposing various ideas and achitectures (GANs, Transformers and Diffusion models) on how to improve the quality and consistency of the generated sequences. Bellow we will elaborate on some notable approaches.

3.1 StoryGAN

StoryGAN was the first model to be proposed for the task of Story Visualization. An overview of the model can be seen Figure 3.1.1. In terms of notation, let us denote $S = [s_1, s_2, ..., s_T]$ the sequence of sentences that form a story, $X = [x_1, x_2, ..., x_T]$ the corresponding ground-truth images and $\hat{X} = [\hat{x_1}, \hat{x_2}, ..., \hat{x_T}]$ the corresponding generated images.

As the name implies, StoryGAN is implemented as a GAN. The main components of the model are:

- A Story Encoder
- A two-layer Recurrent Neural Network (RNN) based Context Encoder
- An Image Generator
- An Image Discriminator
- A Story Discriminator

3.1.1 Story Encoder

The Story Encoder is represented by the dashed pink box in Figure 3.1.1. It learns a mapping from Story S to a low dimensional embedding vector h_0, which serves as the initial hidden state of the RNN Context Encoder. The Story Encoder $E(.)$ samples the vector h_0 from a normal distribution $h_0 \sim E(S) = \mathcal{N}(\mu(S), \Sigma(S))$. Both $\mu(.)$ and $\Sigma(.)$ are implemented as Multi-Layer Perceptrons. $\Sigma(S) = diag(\sigma^2(S))$ is restricted to a diagonal matrix. The Story Encoding can then be written as $h_0 = \mu(S) + \sigma^2(S)^{\frac{1}{2}} \odot \epsilon_S$, where $\epsilon_S \sim \mathcal{N}(0, 1)$.

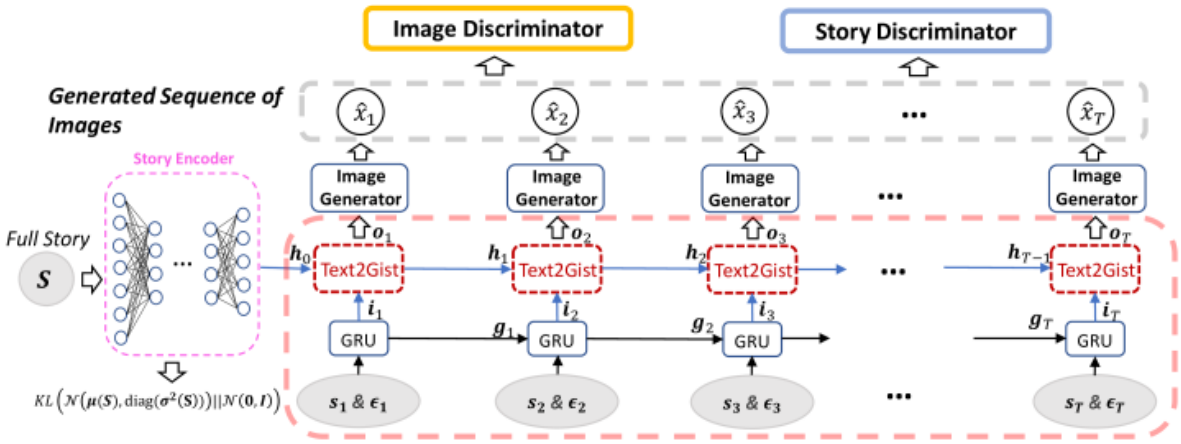


Figure 3.1.1: An overview of the architecture of StoryGAN [18]

3.1.2 RNN Context Encoder

The Context Encoder is depicted in red dashed box in Figure 3.1.1. The word context here refers to any information in the story that is useful for the image that is currently generated. The RNN Context Encoder consists of two layers. The lower layer used standard GRU cells. The second layer employs a variant of the standard GRU, named Text2Gist, that was proposed in the paper.

At time step t, the GRU layer takes in the sentence s_t concatenated with isometric Gaussian noise ϵ_t, and outputs the vector i_t. Then, the Text2Gist takes in i_t and combines it with the story context h_t (initialized by Story Encoder) to generate o_t that will serve as input to the image generator. h_t is updated by the Text2Gist cell to reflect the change of potential context information. In contrast to standard GRU cells that output a vector, Text2Gist transforms i_t into the multi-channel filter o_t of size $C_{out} \times 1 \times 1 \times len(h_t)$, where C_{out} is the number of channels. This is suitable to be used as input to the Convolutional Image Generator.

3.1.3 Image Generator

The Image Generator is a fully Convolutional Network, of no particular interest. It employs a cascade of convolutional blocks, each of them followed by an upsampling block, in order to transform the $C_{out} \times 1 \times 1 \times len(h_t)$-sized input into a $3 \times 64 \times 64$ image.

3.1.4 Image Discriminator

The Image Discriminator is responsible for local consistency. It measures whether the generated image $\hat{x}_t$ matches the corresponding sentence s_t. It does so by learning to discriminate between fake triplets $\{s_t, h_0, \hat{x}_t\}$ and real triplets $\{s_t, h_0, x_t\}$. The image Discriminator is implemented as a Convolutional Neural Network

3.1.5 Story Discriminator

The Story Discriminator is responsible for the global consistency of the generated sequence of images. Its architecture can be seen in Figure 3.1.2. The left part is an image encoder that encodes images (either real or generated) into a sequence of vectors, that are concatenated into a single big vector (depicted in blue). Similarly, the right part is a text encoder that maps the sentences is story S into a sequence of vectors that are then concatenated into the red vector. The red and the blue vector are then multiplied element-wise and the result is used to obtain a score between 0 and 1 that represents the text-image compatibility. The discriminator is, of course trained to distinguish between real and fake image-text sequences.

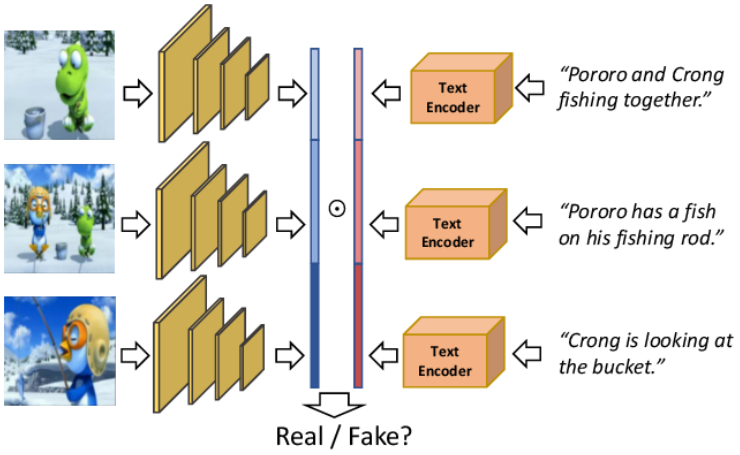


Figure 3.1.2: The Story Discriminator [18]

3.2 CP-CSV

An overview of the architecture of CP-CSV (Character Preserving Coherent Story Visualization) can be seen in Figure 3.2.1. Heavily influenced by storyGAN, CP-CSV adopts the exact same architecture for the Story Encoder, that implements a Normal Distribution by learning its Mean and Variance as two seperate Neural Networks, contitioned on the Story input. The model also adopts StoryGAN's Text Encoder, based on a two-level RNN, utilizing the tailor-made Text2Gist cells in the second layer.

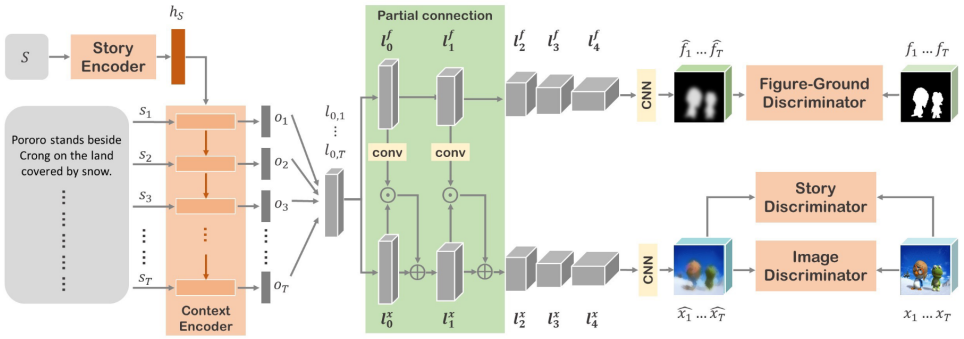


Figure 3.2.1: An overview of the architecture of CP-CSV [36]

The most notable idea introduced in CP-CSV is the utilization of Foreground-Background segmentation for Story Visualization. Specifically, the authors of the paper argue that a logical path to image quality improvement is through the improvement of the appearance of specific characters that appear in the images, and their actions. These characters usually appear on the foreground of most images. Based on this observation, they add an auxiliary foreground-segmentation module, to the network, which learns to predict the foreground (character region) of the image vs its background.

The foreground module is implemented as an extra Image Generator, additional to the one used in StoryGAN. The conventional Image Generator that predicts the images of the story can be seen in the lower half of the green region in Figure 3.2.1, while the image foreground generator can be seen in the upper-half. Both Generators are conditioned on the outputs of the RNN Text-Encoder, exactly as in StoryGAN.

In order to utilize the Foreground-Background information of the auxiliary module, during the image generation process of the primary Generator, CP-CSV exploits Partial Connections between the two Generators (they can be seen inside the green box in Figure 3.2.1, in the space between the Foreground Generator and

the Image Generator). Specifically, features from the Foreground Segmentation model are in the $k-th$ layer denoted as l_k^f are first projected to the space of image features (denoted as l_k^x) in the corresponding layer through a Convolution F_f. Then they are multiplied with the image features to aid with image generation:

$$p_k^f = F_f(l_k^f) \tag{3.2.1}$$

$$l_k^x = l_k^x * p_k^f + l_k^x \tag{3.2.2}$$

Naturally, since the whole model operates withing a GAN framework, an extra Foreground-Background Discriminator is needed to help the corresponding generator learn to predict Foreground-Background maps. Similarly, to the Image Discriminator in StoryGAN, this module learns to distinguish between real and fake segmentation maps based on the sentence and story input.

The Image Discriminator and Story Discriminator are held the same as in StoryGAN.

3.3 DUCO-StoryGAN

DUCO-StoryGAN (Dual Learning, Copy-Transform StoryGAN) [20], adopts StoryGAN as its backbone, leaving several components untouched. Specifically, it adopts the Story Encoder, the Image Discriminator and the Story Discriminator. The full arhitecture of the model can be seen in Figure 3.3.1. The main contributions are the following:

- The introduction of Dual Learning through video redescription for better alignment of the generated images with the input story.
- The introduction of a Copy-Transform mechanism for improved sequential consistency between images.
- Utilization of a Transformer model for more expressive text encoding.

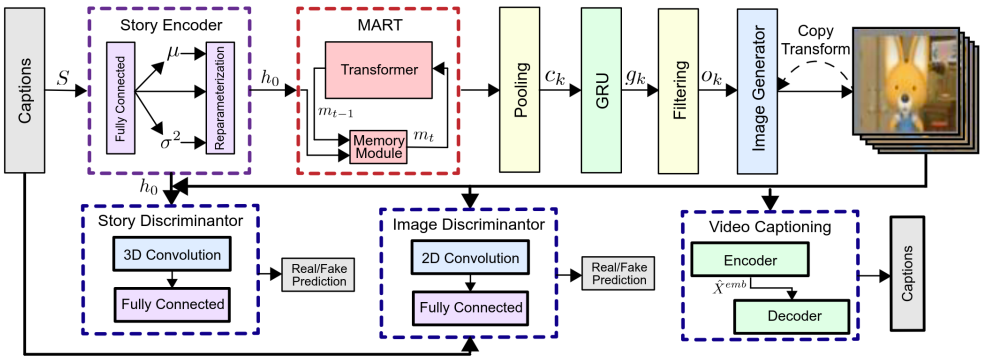


Figure 3.3.1: An overview of the architecture of DUCO-StoryGAN [20]

3.3.1 Mart Context Encoder

The authors of the paper design a Recurrent Context Encoder, in order to maintain consistent background imagery and character appearances throughout the stories. The central module in the Context Encoder is the Memory Augmented Recurrent Transformer (MART) [15]. The memory is initiliazed with the vector h_0 from the Story Encoder. MART takes in word embeddings $W_k = [w_{k1}, w_{k2}, ..., w_{kL}]$, corresponding to the image caption at each timestep k. It outputs contextualized embeddings, that are then pooled into a single weighted representation c_k using an attention mechanism. This procedure is supposed to let the encoder capture interactions between words in an expressive way. The output c_k is then passed into a GRU layer and transformed into a filter to be passed into the Image Generator.

3.3.2 Dual learning via Video Redescription

Dual Learning adds an extra learning signal, based on the following observation, about the duality of out task: if the story captions S can be used to produce the story images $\hat{X}$, then $\hat{X}$ can be used to produce S. The forward task is Story Visualization, while the reverse one can be referred to as Video Rediscription. The authors of the paper leverage a Video Captioning Network that they train on ground truth data and then freeze its weights. This is then used when training DUCO; the generated images are fed into the Video Captioning Network to produce captions. The produced captions are then compared to the ground truth ones to provide extra feedback for the image generation process.

3.3.3 Sequentially Consistent Story Visualization: Copy-Transform

The authors of the paper point out that several components of images such as background and character appearances are largely preserved between frames of the same story. To take advantage of this observation, a Copy-Transform mechanism is introduced. The Copy-Transform module performs word-image attention between text features of the current timestep and image-features of the previous timestep. The generated attention scores are used to produce a weighted version of image features from the previous timestep, to

concatenate with the current image features. The combined image features are used by the Image Generator to produce a context-aware image at every timestep.

3.4 VLC-StoryGAN

As its name implies, VLC-StoryGAN (Visuo-spatial, Linguistic and Commonsense StoryGAN) [19] is yet another GAN developed for Story Visualization based on the Original StoryGAN architecture. The new ideas proposed in the VLC-StoryGAN paper can be summarized as follows:

- Using constituency parse trees and commonsense knowledge to improve language encoding
- Leveraging extra additional semantic and positional feedback through a pretrained dense captioning model
- Aiding the model to recognise subtle intra-story changes by training with intra-story constrastive loss.

An overview of the model's architecture can be seen in Figure 3.4.1. Subsequently we will analyze its components, that differ from its StoryGAN-based predecessors.

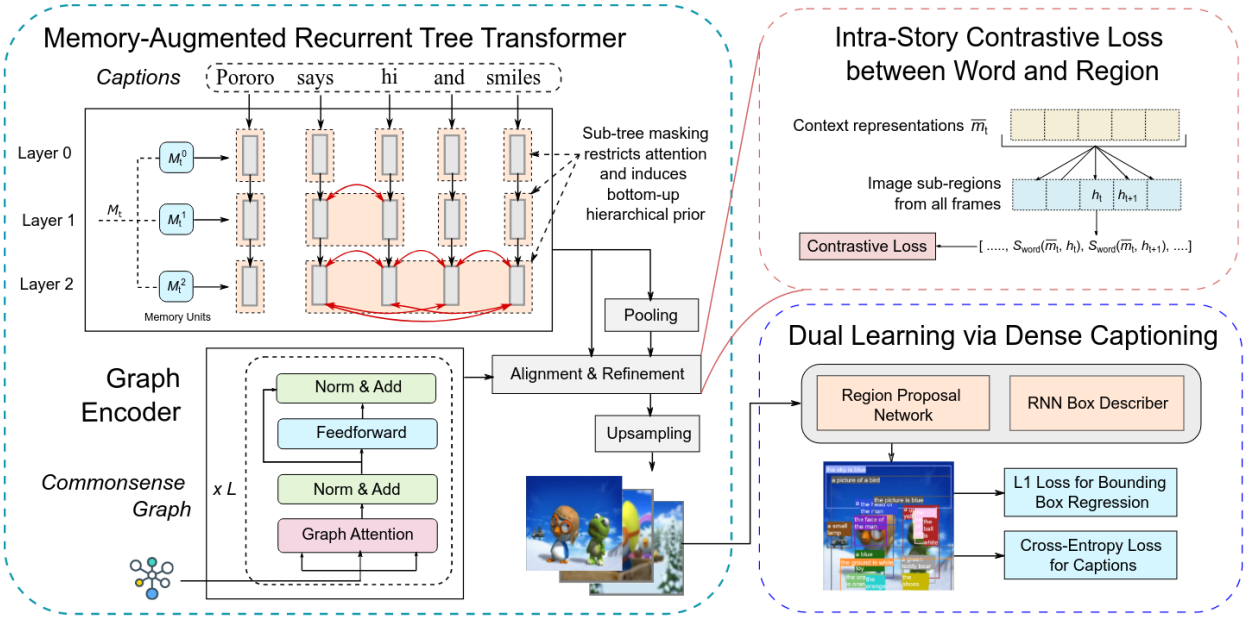


Figure 3.4.1: An overview of the architecture of VLC-StoryGAN [19]

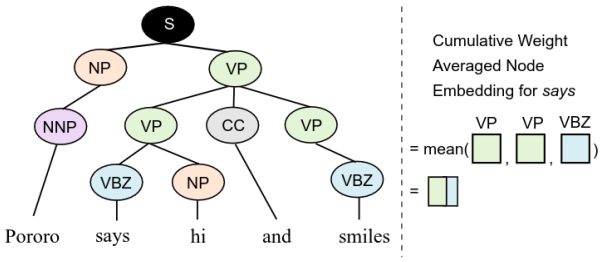


Figure 3.4.2: Example of a constituency parse tree [19]

3.4.1 Memory-Augmented Recurrent Tree Transformer

For Text Encoding, a novel module is proposed by the authors of the paper, named Memory-Augmented recurrent Tree Transformer (MARTT). Based on the original MART, it used constituency parse trees to better take into account the hierarchical structure of sentences. Constituency Parse trees are grammatical

structures that analyze a sentence in a tree-like manner consisting of different types of nodes. Specifically, a Sentence S is recursively broken into Noun Phrases (NP) and Verb Phrases (VP) until each node represents a single word. An example of such a Tree can be seen in Figure 3.4.2.

When encoding a sentence S of length n (an image caption), it is first passed through a parser to produce its constituency parse tree $G(S)$. $T(S)$ denotes the ordered sequence of n terminal nodes (or leaves) of the tree and $N(S)$ denotes the set of non-terminal nodes (or simply nodes), each of which has a phrase label (e.g., NP, VP) and spans over a sequence of terminal nodes. The embeddings of leaf nodes are the concatenation of a word embedding and the corresponding node embedding. Node embeddings are computed as the upward cumulative average over the nodes that the respective non-terminal leaf token is a child of. For example, as we can see in Figure 3.4.2, the Node embedding for *says* is the average of all non-terminal nodes in the upward path from *says* to the root. None Representations are learned during training.

The resulting leaf embeddings for a sentence are used as input for the Transformer (MARTT). Inside the Transformer, in the self-attention layers, sub-tree masking is employed instead of standard causal masking. Specifically, for each word-query, the attention mechanism only has access to the other members of the sub tree (of the sentences constituency parse tree) at that layer. For example, in Figure 3.4.1, in the first Transformer layer, every token only attends to itself. In the second layer, *says* and *hi* can attend to each other, since they belong to the same sub-tree at that level and so forth.

This complicated bottom-up approach in the Text Encoding process is supposed to let the model better understand and encode the hierachical structure of language and the relationships between words.

3.4.2 Commonsense Knowledge

The authors of the paper point out that the image captions in popular Story Visualization datasets frequently omit information that can be considered to be redundant and commonsense for humans. For example, in a scene where the characters are standing outside on a sunny day, the blue sky, the bright sun and the type of lighting is not necessarily described in the caption. However, such information should not be considered redundant for a neural network. In order to enrich the descriptions with such knowledge, the researchers extract commonsense concepts relevant to each frame using a entity-relationship method [2] on ConceptNet. The commonsense knowledge paths are combined into a subgraph and encoded using a Graph Transformer. Finally, the resulting encodings are combined with the outputs of MARTT, as we can see in the leftmost dashed box in Figure 3.4.1.

Dual Learning with Dense Captioning

A pretrained Network is used as a reference for dual learning, similarly to what was done in DUCO-StoryGAN. However, here, a dense captioning model is used. The model takes in images and outputs multiple bounding boxes and corresponding descriptions of what is taking place inside each box. By comparing the outputs for ground-truth and generated images, an extra visuo-spatial (bounding boxes) and semantic (bbox descriptions) feedback is provided to the model.

3.4.3 Contrastive Loss

At first, a pairwise cosine similarity between all image-subregions and word tokens at the current timesteps is computed and based on them, a word-context vector a_j is computed for the j^{th} subregion of the image. Then, an alignment score between all subregions h_k if image x_k and all words in the caption s_k is computed as follows: $S_{word}(x_k, s_k) = \mathbf{log}(\sum_{j=1}^{N} exp(cos(h_{jk}, a_{jk})))$, where N is the number of image subregions. Finally, the intra-story contrastive loss is given by the following formula:

$$\mathcal{L}_{word} = -log\frac{exp(S_{word}(x_k, s_k))}{\sum_{m=1}^{T} exp(S_{word}(x_m, x_k))} \tag{3.4.1}$$

where T is the number of images in the story. This loss term attempts to force the model to take into account subtle differences between frames in the same story, by penalizing similarity between captions and generated images from different timesteps.

3.5 VP-CSV

VP-CSV (Visual Planning for Character-Centric Story Visualization) [5] is the first work on Story Visualization to depart from the GAN framework. VP-CSV is based on text-to-image framework that has shown promising results, consisting of a VQ-VAE that learns a mapping to a discrete latent space for images and a Language Transformer that learns to predict the image latents conditioning on text inputs. The original text-to-image technique is analyzed in depth in the later sections of this thesis. The authors of the paper adapt the method to the task of Story Visualization by breaking the Language Transformer stage into two separate stages. The first one uses a Transformer to predict visual tokens that correspond to the characters in the stories, thus focusing on Character Generation. The final stage employs a second Transformers that completes the latent representations from the previous stage with missing background tokens. The final tokens can then be decoded into an image, using the VQ-VAE's decoder.

3.5.1 VQ-VAE

The VQ-VAE [39] can be viewed as an Encoder-Decoder architecture. The Encoder and the Decoder, are both Convolutional Neural Networks (CNNs). In between the two networks a Vector Quantization (VQ) layer is introduced.

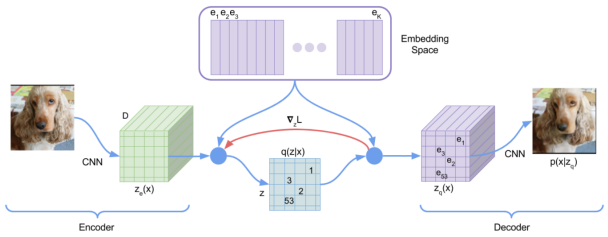


Figure 3.5.1: Overview of the VQ-VAE architecture [39]

The **Encoder** learns to map an $N \times N \times 3$ image into a latent representation $\in \frac{N}{f} \times \frac{N}{f} \times D$, where f is the downsampling factor and D is the number of channels. The **VQ** layer has an vector library of D-dimensional embeddings $e \in \mathbf{R}^{K \times D}$, where K is the size of the library (number of embeddings). When passed through the VQ layer, the output of the Encoder is Quantized in the following manner: every one of the $(\frac{N}{f} \times \frac{N}{f})$ D-dimensional vectors in the latent representation is substituted by its closest vector in the embedding library according to the euclidean distance. Finally, the **Decoder** learns to map the $(\frac{N}{f} \times \frac{N}{f} \times D)$-sized quantized latent representation that is outputed by the VQ layer back into $\mathbf{R}^{N \times N \times 3}$, i.e. back into image space.

A much more detailed description of VQ-VAE's architecture can be found in Chapter 5.

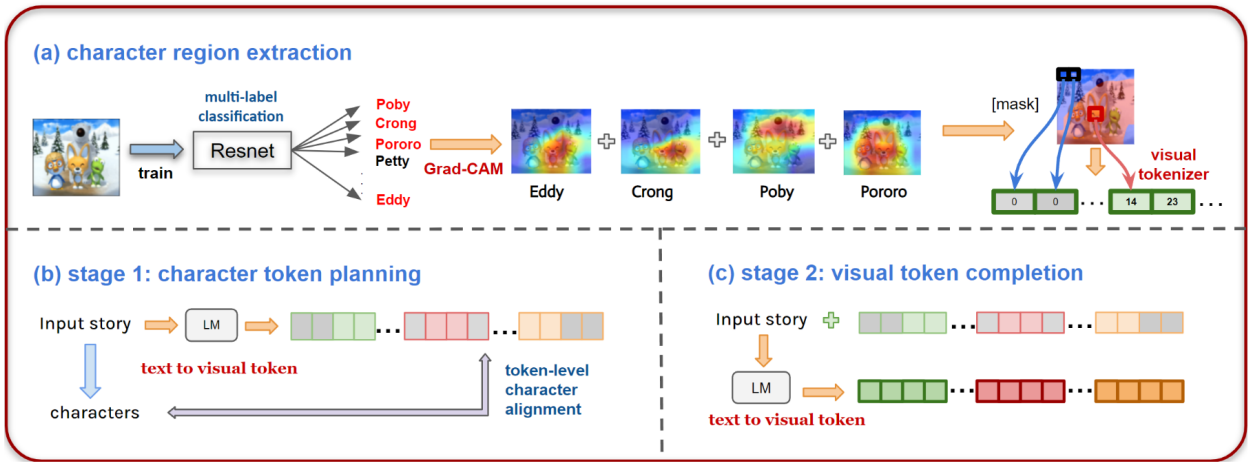


Figure 3.5.2: Overview of the architecture of VP-CSV [5]

3.5.2 Visual Planning (VP)

As we have already hinted, the Visual Planning stage attempts to generate a character visual plan, to determine where the main characters should be placed in each image, before filling in background information. Since there are no annotations for character regions in the task's datasets, the researchers use an image segmentation technique to extract these regions.

Character Region Extraction

At first, a multilabel classifier CNN is trained to recognize the presence of the 9 main characters in images. GradCAM [31] is then used to extract heatmaps that highlight image regions that are highly attended by the classifier when deciding whether a character is present in the image. For each image, the heatmaps for all characters that are present in the caption are merged into one. If an image region is not activated in any of the heatmaps, it is considered to be part of the background and therefore masked out. A visual example of this process can be seen on the upper half of Figure 3.5.2.

Character Token Planning (First Stage)

A GPT-2 model is trained to produce the Character Visual Tokens. Specifically, it conditions on the input story captions $S = \{s_1, s_2, ..., s_n\}$ and learns to predict the Character Tokens $R = \{r_1, r_2, ..., r_n\}$, that have been computed offline for the training set, as described in the previous section (n is the number of frames per story). The loss function under which the Transformer is trained is:

$$\mathcal{L}_\theta = -\mathbf{log}p(r|s, \theta) \tag{3.5.1}$$

where θ represents the Transformer's parameters.

Visual Token Completion (Second Stage)

In the second stage, another GPT-2 model is trained to predict all tokens conditioning both the input stories and the Character Tokens produced in the previous stage. The loss function for this stage can be expressed as follows:

$$\mathcal{L}_\theta = -\mathbf{log}p(z|s, r, \theta) \tag{3.5.2}$$

where z represents the complete latent representation (all visual tokens). The Transformer model in this stage is initialized with the weights of the previous stage, to reinforce attention to the character tokens.

3.5.3 Token Level Character Alignment

The authors of the paper propose an additional way to improve character generation. At first they compute the visual token distribution for each character (The distribution of latent visual tokens when encoding the character). Then, they extract the 10 most frequent tokens t_c for each character. The utilization of these tokens is then reinforced, by using an extra semantic loss term that biases the model towards adopting them when generating the corresponding character.

3.6 CMOTA

CMOTA (Context Memory and Online Text Augmentation) [1] builds on the text-to-image VQ-VAE/Transformer architecture, like VP-CSV. In short, the paper makes the following contributions:

- Using a special memory module to encode and propagate context between different frames in the same story
- Training the Transformer model to predict either Image from Text (i2t) or Text from Image (t2i). This way pseudo-texts can be generated as extra descriptions for images, to enrich the training dataset with additional captions

3.6.1 Base Model

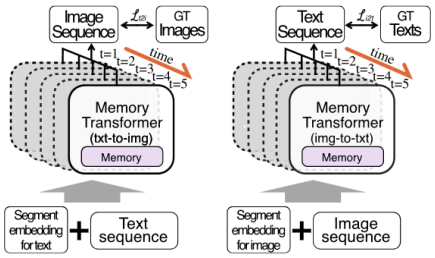


Figure 3.6.1: CMOTA's Bidirectional Transformer [1]

The Transformer iteratively generates images and texts in both ways. The image tokens are predicted sequentially from input text. In the "reverse" manner, the text tokens are sequentially predicted from the input image tokens. A positional embedding is used, to indicate absolute position in the sequence. Additionally, a segment embedding is introduced to distinguish between source and target modality. Denoting $t = \{t_1, t_2, ..., t_m\}$ and $z = \{z_1, z_2, ..., z_n\}$, the text and image tokens respectively, the loss function $\mathcal{L}_j$ can be written as follows:

$$\begin{aligned} \mathcal{L}_{j,t2i} &= \sum_{k=1}^{n} -\ln p_j(z_k | t_1, ..., t_m, z_1, ..., z_{k-1}) \\ \mathcal{L}_{j,i2t} &= \sum_{k=1}^{n} -\ln p_j(t_k | z_1, ..., z_n, t_1, ..., t_{k-1}) \\ \mathcal{L}_j &= \mathcal{L}_{j,t2i} + \lambda_1 \mathcal{L}_{j,i2t} \end{aligned} \tag{3.6.1}$$

3.6.2 Context Memory

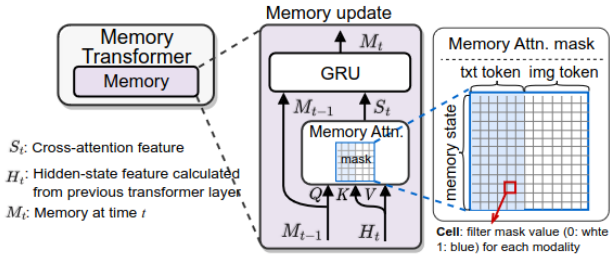


Figure 3.6.2: CMOTA's memory module [1]

Figure 3.6.2 shows how the memory state at timestep t (M_t) is obtained. At first, cross-attention is computed between the current hidden state, $H_t^l \in \mathbb{R}^{T_c \times d}$ and the previous memory state $M_{t-1} \in \mathbb{R}^{T_m \times d}$. ($T_c$: number of Tokens, T_M: number of memory states, d: hidden state dimension). Specifically, the calculation made is: $S_T = attn(M_{t-1}, H_t, H, t)$, where $attn(.)$ stands for the standard Scaled Dot-Product attention. As Figure 3.6.2 shows, a special attention mask is used that only allows attention over the text tokens and prohibits attention over image tokens. This choice is made to disallow strong constraint that could be caused by image token attention.

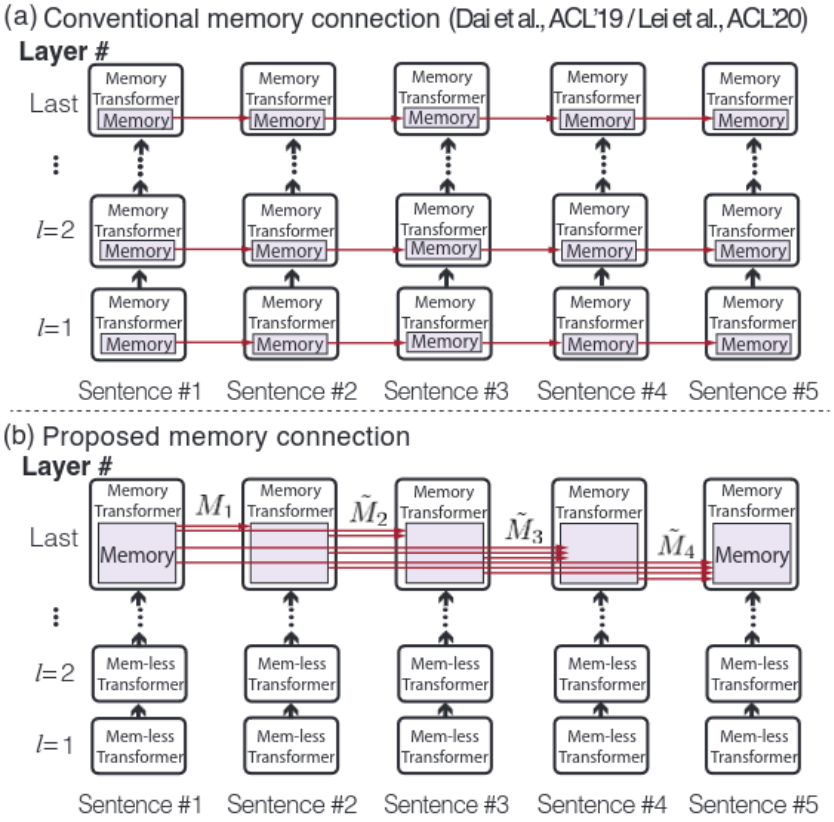


Figure 3.6.3: Comparison between the traditional memory connection scheme and CMOTA's [1]

Figure 3.6.3 (a) shows the traditional memory connection scheme in Transformer models, where there are memory connections in all Transformer layers and only from each timestep to the immediately next one. The researchers behind CMOTA opt for an alternative scheme, where memory is used only in the final Transformer layer. Additionally, the memory module in each timestep has access to all previous memory states, instead of only accessing the immediately previous one (3.6.3 (b)). In addition, the researchers agrue that not all historic information matters equally and therefore adjust the computations to attentively weigh past information as follows:

$$\begin{aligned} \bar{M}_{1:(t-1)} &= Attn(M_{(t-1)}, M_{[1:(t-2)]}, M_{[1:(t-2)]}) \\ \tilde{M}_{(t-1)} &= [M_{t-1}; \bar{M}_{1:(t-1)}], (3 \leq t \leq 5) \\ H_t^l &= Attn(H_t^l, [H_t^l; \tilde{M}_{(t-1)}], [H_t^l; \tilde{M}_{(t-1)}]) \end{aligned} \tag{3.6.2}$$

$M_{(t-1)}$ is the memory at time (t-1) and [1:(t-2)] refers to the concatenation from time 1 to (t-2). At t=2 M_1 is used instead of $\tilde{M}_1$, since the latter cannot be computed.

3.6.3 Online Text Augmentation

The idea behind text augmentation is to address language variations in captions and to bridge this gap during inference. The multi-modal Transformer that works in two directions (*text-to-image* and *image-to-text*) offers a natural way to integrate text augmentation in the training process. Specifically, the pseudo-texts that can be generated using the *image-to-text* direction of the Transformer are adopted as supplementary descriptions for the images. Then, they can be used as input to train the Transformer in the *text-to-image* direction. In early epochs, less meaningful sentences are generated, but with the progress of training much better-aligned

descriptions are provided by the model. To address this, a pseudo-text is tested online, during training to check if it contains a certain percentage of character name references compared to the ground-truth. If it does, it can be used as a caption during training. When using pseudo-texts, the loss function is adjusted as follows:

$$\begin{aligned} \mathcal{L}_{j,pt2i} &= \sum_{k=1}^{n} -\mathbf{ln} p_j(z_k|\hat{t}_1, ..., \hat{t}_m, z_1, ..., z_{k-1}) \\ \mathcal{L}_j &= \mathcal{L}_{j,t2i} + \lambda_1 \mathcal{L}_{j,i2t} + \lambda_2 \mathcal{L}_{j,pt2i} \end{aligned} \tag{3.6.3}$$

where $\mathcal{L}_{j,pt2i}$ represents the additional loss with the augmented pseudo-texts and $\hat{t}$ represents a pseudo-text.

3.7 AR-LDM

AR-LDM (Auto-Regressive Latent Diffusion Model) [22] is-to our knowledge-the first work to use a diffusion model for Story Visualization. Specifically, it enriches a text-to-image diffusion model, with a history-aware module, that helps maintain story context.

3.7.1 Diffusion Models

For the shake of completeness, we deem it necessary to make a brief reference on how Diffusion works, before moving on. Diffusion Models [34] define a Stochastic Process (Markov Chain) q that gradually adds Gaussian Noise to a sample of real data $z_0 \sim q(z)$ in T steps. The data, we are sampling (z) could be images. However, in the case of Latent Diffusion, z denotes a latent representation of an image obtained from a VAE (VAEs are extensively covered in Chapter 5). The forward process at each step can be defined as follows:

$$\begin{aligned} q(z_t|z_{t-1}) &= \mathcal{N}(z_t; \sqrt{1-\beta_t} z_{t-1}, \beta_t \mathbf{I}) \\ q(z_{1:T}|z_0) &= \prod_{t=1}^{T} q(z_t|z_{t-1}) \end{aligned} \tag{3.7.1}$$

where $\beta_t \in (0, 1)$ is the step size.

Diffusion Models (usually UNets [29]) are trained to learn a step-by-step reversion of the forward process described above. This way they can derive real world Data (e.g. images) from noise. We define the following:

$$\begin{aligned} \alpha_t &= 1 - \beta_t \\ \bar{\alpha}_t &= \prod_{i=1}^{t} \alpha_i \end{aligned} \tag{3.7.2}$$

Then the denoising process $p(.)$ can be parameterized as follows:

$$\begin{aligned} &p_\theta(z_{t-1}|z_t) = \mathcal{N}(z_{t-1}; \mu_\theta(z_t, t), \Sigma_\theta(z_t, t)) \\ &\text{where } \mu_\theta(z_t, t) = \frac{1}{\sqrt{\alpha_t}}(z_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(z_t, t)) \end{aligned} \tag{3.7.3}$$

Specifically, the UNet is trained to learn ϵ_θ.

3.7.2 The architecture of AR-LDM

An overview of the model's architecture can be seen in Figure 3.7.1. The yellow boxes represent the Diffusion Model. The green and purple boxes (BLIP and CLIP models) form the history conditioning model. Finally, the blue boxes marked with $\mathcal{E}$ and $\mathcal{D}$ represent the Encoder and the Decoder of the VQ-VAE (see Chapter 5) that allows the model to work in a latent space, instead of the pixel space.

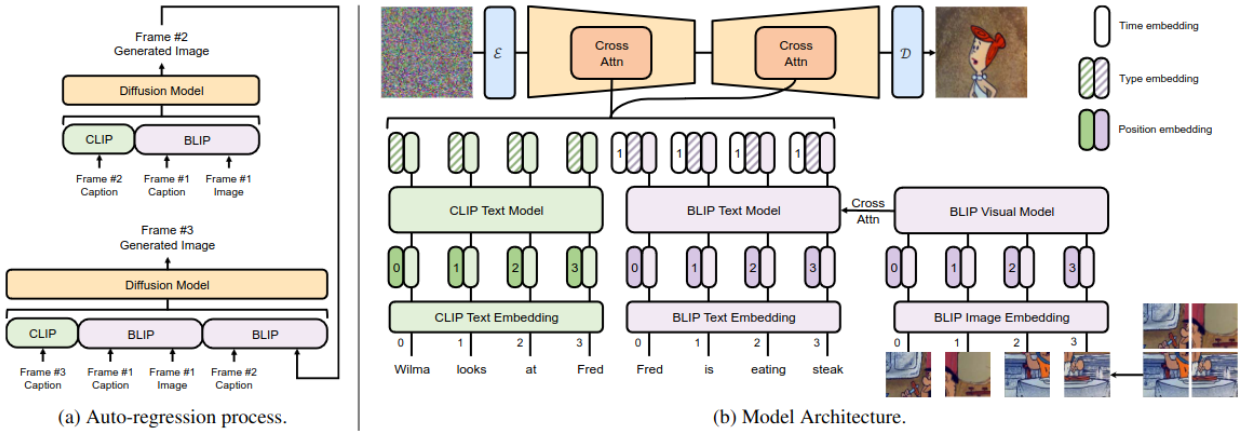


Figure 3.7.1: Overview of AR-LDM's architecture [22]

Auto-Regressive Latent Diffusion Model

A standard diffusion approach for *text-to-image* would condition the diffusion process on a single caption to generate the image. AR-LDM departs from this approach by conditioning the generation of a frame on all previous captions and all previously generated frames, in a auto-regressive manner. This is supposed to help generate a more coherent storyboard or images. For a story of length L we denote $C = [c_1, ..., c_L]$, $X = [x_1, ..., x_L]$ and $\hat{X} = [\hat{x}_1, ..., \hat{x}_L]$ the captions, ground-truth images and generated images. The model, then estimates the posterior probability as follows:

$$\begin{aligned} P_{AR}(X|C) &= \prod_{j=1}^{L} P(x_j|\hat{x}_{<j}, C) \\ &= \prod_{j=1}^{L} P(x_j|\tau_\theta(\hat{x}_{<j}, c_{\leq j})) \\ &= \prod_{j=1}^{L} p_\theta(z_0^{[j]}|\tau_\theta(\mathcal{D}(z_0^{[<j]}), c_{\leq j})) \end{aligned} \tag{3.7.4}$$

where p_θ is the reverse diffusion process, described in the previous section and τ_θ is the history-aware conditioning network.

Generative Network

As we have already mentioned, the generative model performs the forward and backward diffusion processes in a latent space instead of the pixel space, following [28]. This is done for the sake of efficiency, since the latent space is more compressed. In order to work in the latent space, an Autoencoder, consisting of an Encoder $\mathcal{E}$ and a Decoder $\mathcal{D}$) is used. The Autoencoder is trained to reach a point where $\mathcal{D}(\mathcal{E}(x)) \approx x$ holds, for an image x. Diffusion works with representations $z = \mathcal{E}(x)$.

History-Aware Conditioning Network

The Conditioning Network encodes historical information from captions and previously generated frames into a multimodal condition $\phi_j = \tau_\theta(\hat{x}_{<j}, c_{\leq j})$. Based on this, $P(x_j|\hat{x}_{<j}, C)$ in Equation 3.7.4 can be rewritten as $p_\theta(z_0^{[j]}|\phi_j)$. The Conditioning Network leverages CLIP [23] and BLIP [17]. CLIP is charge of encoding the current caption in a unimodal way. On the other hand, BLIP uses cross-attention between text and vision modalities to integrate text and visual features from previously generated frames and the corresponding captions. The multimodal condition can be written as follows:

$$\begin{aligned} \bar{c}_j &= CLIP(c_j) \\ \bar{m}_{<j} &= [BLIP(c_1, \hat{x}_1), ..., BLIP(c_{j-1}, \hat{x}_{j-1})] \\ \phi_j &= [\bar{c}_j + c^{type}; \bar{m}_{<j} + m^{type} + m^{time}_{<j}] \end{aligned} \tag{3.7.5}$$

where $\bar{m}_{<j}$ denotes multimodal encoded features from previous captions and frames. c^{type} and m^{type} are text and multimodal type embeddings, respectively. m^{time} is the time embedding.

Experimental Settings

It is reported in the paper that the diffusion model is initialized with the weights of stable diffusion [28], a model that has been trained on the LAION-5B dataset. AR-LDM is reportedly trained for 50 epochs on 8 NVIDIA A100 GPUs (80 GB each) for 2 days.

3.8 ACM-VSG

ACM-VSG[10] is a diffusion-based approach similar to AR-LDM. It leverages pre-trained Stable-Diffusion[28] as its base and enhances it with cross-attention mechanisms. As in AR-LDM, the image sequence generation is modeled in an auto-regressive manner. Cross-attention allows the model to integrate multi-modal features from previous caption-frame pairs into the currently generated frame, to improve consistency. Additionaly, an Adaptive Guidance mechanism is introduced, that explicitly pushes frames that have similar captions, to be similar as well.

3.9 Causal-Story

Causal-Story[35] is closely related to the the two previous diffusion approaches and especially AR-LDM. It improves the cross-attention mechanism used in AR-LDM, by introducing a Local Causal Attention Mask. This way, it limits the size of historical context tokens, thus eliminating confusion, caused by interfering captions.

3.10 Story-LDM

Story-LDM (Story Latent Diffusion Model) [25] is the second work on Story Visualization to leverage diffusion models. In fact, the approach proposed in the paper is quite similar to the one of AR-LDM. It leverages a pretrained diffusion model [28], which is the exact same one that was used for AR-LDM. The diffusion model is enhanced with a special memory attention mechanism. This mechanism makes it possible to condition on the current caption as well as the previous captions and previously generated frames, whilst only keeping information relevant to the current timestep. In order to use this attention mechanism that conditions on past information, the model has to work in an autoregressive manner, where the frames are generated one at a time and the output of the generation process for one frame is fed back into the system, as input for the generation of the next frames.

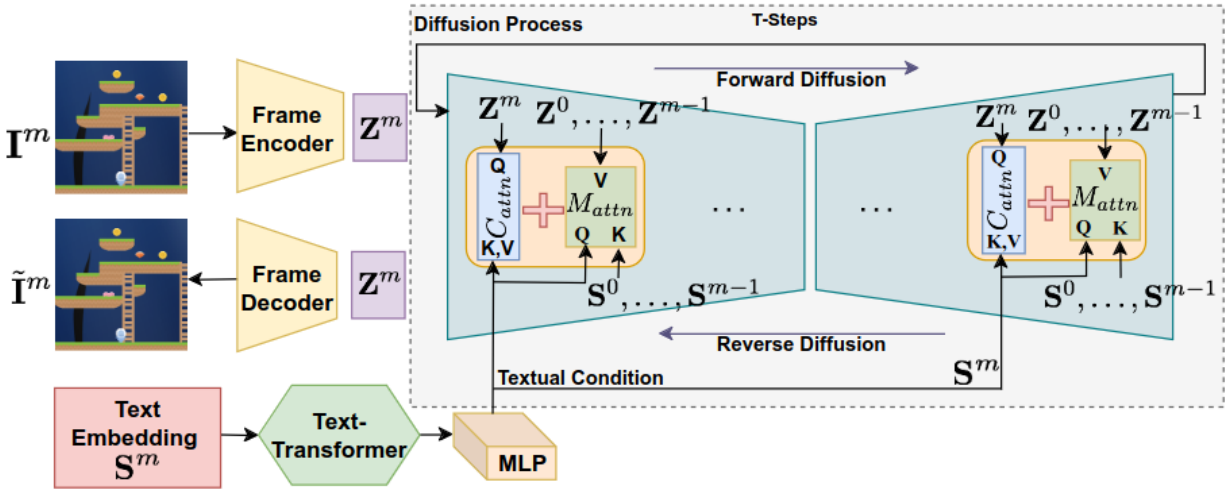


Figure 3.10.1: Overview of Story-LDM's architecture[25]

3.10.1 Latent Diffusion Backbone

As we have already mentioned, the pretrained Diffusion model [28] leveraged by Story-LDM is the exact same one that was used for AR-LDM. A brief account of how it works is given in Section 3.7.1. It trains a UNet to learn ϵ_θ, which parameterizes the reverse diffusion process:

$$\begin{aligned} &p_\theta(z_{t-1}|z_t) = \mathcal{N}(z_{t-1}; \mu_\theta(z_t, t), \Sigma_\theta(z_t, t)) \\ &\text{where } \mu_\theta(z_t, t) = \frac{1}{\sqrt{\alpha_t}}(z_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(z_t, t)) \end{aligned} \tag{3.10.1}$$

z_0, which we want to obtain through the reversal of the diffusion process is the latent representation of an image. The latent space learned by a VQ-VAE.

3.10.2 Story Latent Diffusion Model

For Story Visualization, the researchers extend the vanilla Diffusion Model to function auto-regressively. The auto-regressive approach conditions on the current caption as well as previous captions and generated frames through cross-attention layers. Using a condition y, the cross-attention layer (scaled dot-product attention) is formulated as follows:

$$Attn(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V \tag{3.10.2}$$

where $Q = W_Q \cdot \hat{f}(Z)$, $K = W_K \cdot f(y)$ and $V = W_V \cdot f(y)$. $\hat{f}(Z)$ is an intermediate, flattened representation of Z withing the diffusion model (Z is the VQ-VAE encoding of the image). $f(y)$ is the feature representation of the condition y. Story-LDM uses cross-attention layers, where the condition y is the text description of the frame that is being generated.

There is a more complete account of how Attention works in Chapter 4.

3.10.3 Memory-Attention Module

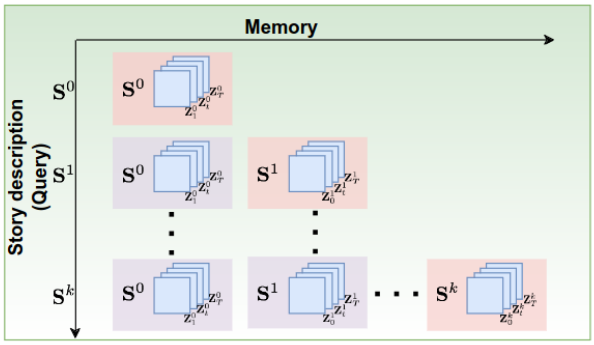


Figure 3.10.2: Overview of Story-LDM's memory-attention module[25]

The Memory-Attention module allows the model to condition on all past descriptions S^i and previously generated frames $\hat{Z}^i$, where $i \in \{0, ..., m-1\}$, when generating the m-th frame. To achieve this, another attention mechanism $Attn(Q, K, V)$ is used where:

$$\begin{aligned} Q &= W_Q \cdot f(S^m), \\ K &= W_K \cdot f(S^{<m}, \\ V &= W_V \cdot \hat{f}(\hat{Z}^{<m}) \end{aligned} \tag{3.10.3}$$

where $\hat{f}$ aligns the dimensions of values with keys. The Memory-Attention modules weighs the relevance of Q, which depends on the current text description against K, that depends on previous descriptions and applies this on the previous image representation $\hat{Z}^{<m}$. This mechanism is supposed to effectively fuse relevant visuo-textual information from previous timesteps into the generation process, in the current timestep.

3.10.4 Network Architecture

As we mentioned, Story-LDM leverages the UNet from [28]. The approach is modified to work in an auto-regressive setting. On overview of the generation process can be seen in Figure 3.10.1. The latent representation of a frame Z^m is obtained using a the Frame Decoder (of a VQ-VAE). A transformer is used to obtain a representation of the text description S^m. The UNet is then used to model the diffusion process, in T steps. All layers of the UNet are augmented with attention. In an attention sub-layer Cross-Attention $C_{attn} = \sum_i \hat{f}(Z^m)_i f(S^m)_i$ is performed followed by Memory-Attention $M_{attn} = \sum_{k=1}^{m-1} \sum_i \hat{f}(Z^k)_i f(S^k)_i f(S^m)_i$ and the results are aggregated. So, the output of the Attention sub-layer is $C_{attn} + M_{attn}$. Starting from the noise sample Z_T^m the output of the reverse diffusion process Z_0^m is reconstructed using the frame decoder.

We should note that Story-LDM is applied to a modified version of the original task, where repeated references to Character Names are substituted by pronouns (e.g. he, she, they).

3.11 StoryGPT-V

StoryGPT-V [33] is - to our knowledge - the latest work on Story Visualization. It leverages a pretrained Diffusion Model and a pretrained LLM, which are arguably the most prominent model families in Text-to-Image Generation and NLP, respectively, at the moment. StoryGPT-V is devised as a two-stage approach. The first state modifies the pretrained Diffusion Model, to specifically focus on Character Generation. At the second stage, the pretrained LLM is aligned with the Diffuser, to help with the consistency of the generated image-stories.

3.11.1 Character-Aware LDM with attention control

Integrating Visual Features with text conditions

In order to improve character generation, the text descriptions are enhanced with visual features of corresponding characters, and the attention of text conditions is guided, to strongly focus on characters. Let s be a text description, which references K characters that should be present in image $\hat{x}$, $\{x_c^1, ..., x_c^K\}$ be images of those characters and $\{i_c^1, ..., i_c^K\}$ be the list of token indices that show the position of each character name in the description.

CLIP's [23] text ($CLIP_T$) and image ($CLIP_I$) encoders, are used to extract textual and visual features respectively. Then, the text tokens that represent a character's name are augmented with visual information. Specifically, these text tokens are concatenated with the visual features of the corresponding character and processed by an MLP layer. An augmented token in the augmented embedding c of a description s is formed as follows:

$$c_k = MLP(CLIP_T(S[i_c^k]; CLIP_I(x_c^k))) \tag{3.11.1}$$

For token's in c that are not related to characters Vanilla CLIP embeddings are used, whereas for tokens that reference characters, we use token embeddings given by Equation 3.11.1.

Controlling Attention of Text Tokens

In vanilla LDMs (Latent Diffusion Models), there are no restrictions as to whether a latent pixel can be influenced by a text token. The researchers behind StoryGPT-V choose to introduce such restrictions, in order to guide certain pixels to be more influenced by tokens representing character names. To achieve this, they first obtain segmentation masks for each character present in the caption denoted as $\{M_1, ..., M_K\}$. Then, a regularization loss term is introduced to encourage the cross-attention map A_k for character k at the token position i_c^k, to follow the segmentation mask M_k and avoid the region outlined by $\bar{M}_k$, which is irrelevant to the character:

$$\mathcal{L}_{reg} = \frac{1}{K} \sum_{k=1}^{K} (A_k^- - A_k^+) \tag{3.11.2}$$

where:

$$A_k^- = \frac{A_k \odot \bar{M}_k}{\sum_{i,j} (\bar{M}_k)_{ij}}, \; A_k^+ = \frac{A_k \odot M_k}{\sum_{i,j} (M_k)_{ij}} \tag{3.11.3}$$

The minimization of this loss term supposedly reinforces the models attention to relevant areas for each character and discourages the attention to irrelevant areas. The modified LDM's (Char-LDM) function is visualized in Figure 3.11.1 (a).

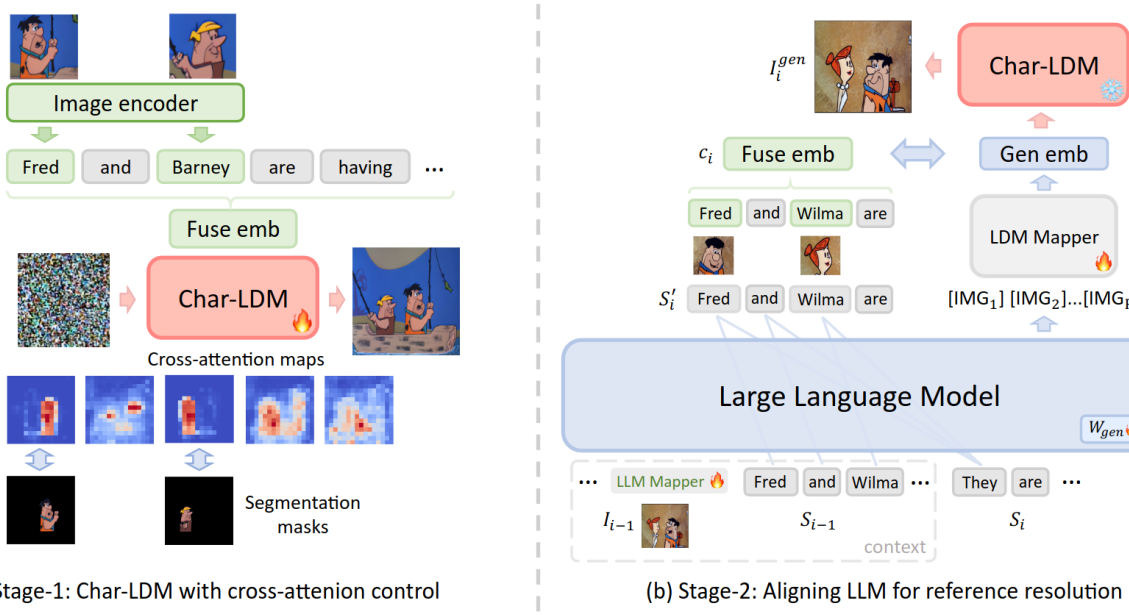


Figure 3.11.1: (a) In the first stage, a fused embedding is created by integrating character visuals with text embeddings, serving as the Char-LDM's conditional input, and the cross-attention maps of Char-LDM will be guided by corresponding character segmentation mask for accurate and high-quality character generation. (b) In the second stage, the LLM takes the interleaved image and text context as input and generates R [IMG] tokens. These tokens are then projected by LDM Mapper into an intermediate output, which will be encouraged to align with fused embedding as Char-LDM's input. The figure intuitively shows how the character-augmented fused embedding and the casual language modeling aid LLM for reference resolution. [33]

3.11.2 Aligning LLM for reference resolution

The first stage, that was described above deals with every caption-image pair on its own, not accounting for the narrative aspect of Story Visualization. StoryGPT-V tackles this challenge by aligning a pretrained LLM, to aid with image coherence and with the disambiguation of referential terms (e.g. he, she, they) that refer to characters. The LLM is trained to autoregressively generate the fused language-visual embeddings that serve as input to the LDM.

Training

The LLM input comprises of interleaved text-descriptions and images. When generating the n-th image, the input is: $(x_1, s_1, ..., x_{n-1}, s_{n-1}, s_n)$, $2 \leq n \leq N$. For the images, visual embeddings are extracted via CLIP: $CLIP_I(x_i) \in \mathbb{R}^{d_i}$ and mapped into the LLM input space using a linear transformation $W_{v2t} \in \mathbb{R}^{d_i \times m \cdot e}$. A visual embedding is mapped into m e-dimensional embeddings, where e is the dimension of the LLM input space. In order to represent visual outputs, R additional tokens $[IMG_1], ..., [IMG_R]$ are introduced, along with a trainable matrix $W_{gen} \in \mathbb{R}^{R \times e}$. The rest of the LLM's parameters are frozen during training.

The training objective that is minimized is:

$$\mathcal{L}_{gen} = -\sum_{r=1}^{R} \textbf{log}p(\ [IMG_r] \mid \mathcal{T}_{prev}, [IMG_{<r}]\ ) \tag{3.11.4}$$

where:

$$\mathcal{T}_{prev} = \{\ CLIP_I(x_{<i})^T W_{v2t}, CLIP_T(s_{1:i})\ \},\ i \in [2, n] \tag{3.11.5}$$

To align the $[IMG]$ tokens produced by the LLM with the LDM input space, a transformer-based function, $Mapper_{LDM}$ is used. An additional loss is computed here, to minimize the distance between the generated embeddings and the augmented text representations that serve as input to the LDM:

$$\mathcal{L}_{align} = || \ Mapper_{LDM}(h_{IMG_{1:R}}) - c_i \ ||_2^2 \tag{3.11.6}$$

Inference

Inference is performed autoregressively. At first, the initial description s_1 is processed. R $[IMG]$ tokens are produced by the LLM and utilized by the Char-LDM to produce the first image $\hat{x}_1$. Subsequently, the LLM uses s_1 and $\hat{x}_1$ as input, along with the second caption s_2 to generate the $[IMG]$ tokens for the second frame. This is repeated until all frames are generated.

As in Story-LDM, StoryGPT-V is applied to a more challenging version of the task, with pronouns in place of repeated Character name references.

Chapter 4

The Transformer

The Transformer is currently the most prominent architecture in the field of NLP. Previous models like RNNs encoded text in a serial fashion, where every text token is produced using a hidden state generated in the previous step. On the contrary, Transformers follow a parallel architecture consisting of cascading blocks that employ attention, a feed-forward network and layer normalization. The original architecture was proposed in 2017 [42] and it has since shown a lot of success first in NLP and more recently in vision as well.

4.1 Original Architecture

The Original Transformer Architecture can be seen in Figure 4.1.1. It consists of two separate networks: the Encoder and the Decoder. the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations z = $(z_1, ..., z_n)$. Given z, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time.

4.1.1 Encoder

The Encoder is composed of 6 cascading identical layers. Each layer comprises of 2 sublayers. The first is a multi-head self-attention layer. The second one is a fully connected feed-forward network. There is a residual connection arround each sublayer, followed by layer normalization. The output of each sublayer can be written as $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ is the core function implemented by the corresponding sublayer.

4.1.2 Decoder

The Decoder comprises of 6 identical layers, similarly to the encoder. Each layer has 3 sublayers. The two of them are same as in the encoder (self-attention and feed-forward network). In between these two sublayers, the decoder inserts an encoder-decoder cross-attention sublayer. Residual connections followed by layer-normalization are used around each sublayer, as in the encoder. Additionally, the self-attention mechanism is modified in order to prevent positions from attending to subequent positions.

4.1.3 Attention Mechanisms

Attention

Attention can be described as a function that maps a query against a set of key-value pairs to produce an output, similarly to what is done in traditional databases. The queries, keys, values and outputs are all vectors. The output is the result of a matrix-vector multiplication. The matrix is computed in way that intuitively represents the compatibility between the query and the corresponding key. This matrix is then multiplied with the corresponding value-vector.

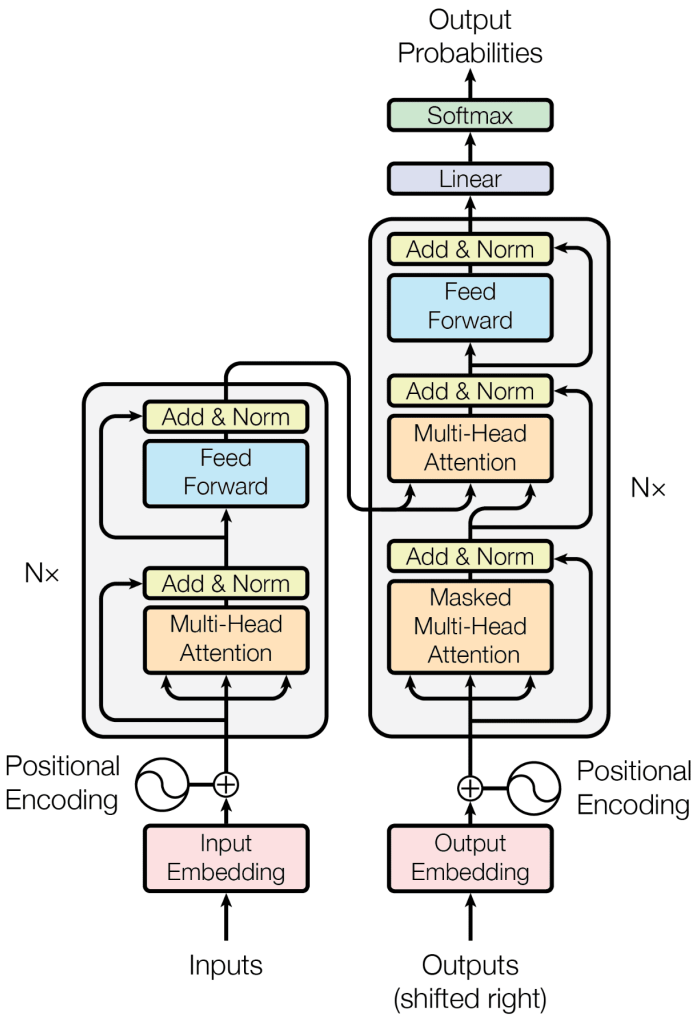


Figure 4.1.1: The original Transformer architecture [42]

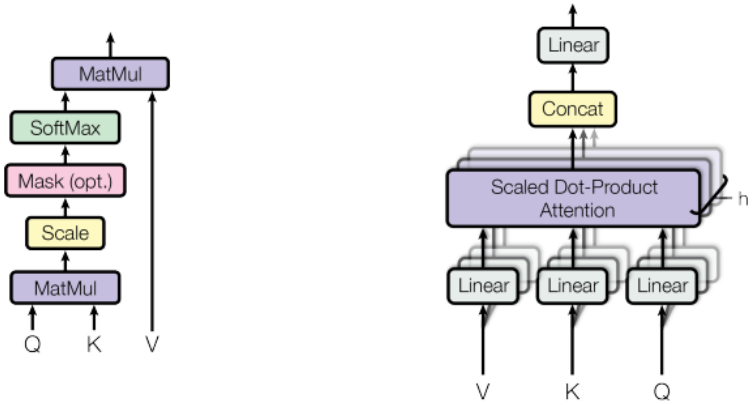


Figure 4.1.2: Scaled Dot-Product Attention (left) and Multi-Head Attention (right) [42]

Scaled Dot-Product Attention

Scaled Dot-Product Attention is the mechanism proposed in the original paper. The computation of the attention function is done for a set of queries packed together in a matrix Q. The keys and values are also packed into matrices K and V, accordingly. The procedure is visually displayed in Figure 4.1.2 (left). As we can see The Query matrix (Q) is multiplied with the Key matrix (K). The output is then scaled and optionally masked (e.g. to prevent a location from attending to future (subsequent) locations). The output is then passed through softmax to essentially obtain a compatibility score matrix. This matrix can be regarded as a stack of vectors, where each vector corresponds to a single Query and represents its compatibility with each Key. The final result is obtained by multiplying this compatibility matrix with the Value matrix (V).

The outputs are formally computed as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4.1.1}$$

Multi-Head Attention

Multi-Head Attention (Figure 4.1.2 (right)) essentially extends the idea of the simple attention, in order to compute multiple attention functions in parallel. The queries, keys and values are linearly projected h times, with different learned linear projections. On each one of these projected version of queries and key-value pairs we compute the attention function, in parallel. The results are then concatenated and and projected once again to get the final result. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O \tag{4.1.2}$$

where:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{4.1.3}$$

where the projections are matrices $W_i^Q \in \mathbf{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbf{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbf{R}^{d_{model} \times d_v}$ and $W_i^O \in \mathbf{R}^{hd_v \times d_{model}}$.

Chapter 5

VQ-VAE

The VQ-VAE (Vector-Quantization Variational Autoencoder) [39] was first introduced in 2018. Like most VAEs, it comprises of two subnetworks: the encoder and the decoder. The encoder is trained to map the original data (images in our case) into a latent representation space (a sequence of vectors). At the same time the decoder learns to reconstuct the original data, based solely on the latent representation provided by the encoder. Thus, the VQ-VAE attempts to learn a compact and expressive representation space for the modality at hand. What sets VQ-VAEs apart from simple VAEs is the fact that their latent representation space is discrete. That is, the sequence of vectors that is outputed by the encoder is quantized based on an embedding library, with a set number of embeddings, before being fed into the decoder. The parameters of the library are learned during training. The discrete nature of the latent space means that we can interpet each vector of the latent representation as a discrete visual token, or interchangeably as the token's index in the embedding library. That makes it possible to then train a transformer to predict the visual tokens with cross-entropy loss.

Intuitively, one can think of a VQ-VAE that learns to encode images as a kind of visual memory mechanism, similar to the one developed by a human artist. Given an image, on the one side, the encoder can map it into an internally meaningful representation using a compact and expressive library of visual tokens, as the eyes of the artist could do. Then, based only on this compact visual memory (representation) the decoder can reconstruct the image preserving most high level features, as an artist could do, drawing an image out of memory.

5.1 Original Architecture

In this section we will take a more formal look at the original VQ-VAE architecture. Traditional VAEs consist of three main parts. The first one is the encoder network that parameterises a posterior distribution $q(z|x)$ of latent random variables z, given the input data (image) x. The second one is a prior distribution $p(z)$. The final one is the decoder network that learns a distribution $p(x|z)$. In the VQ-VAE a discrete latent variable space is used, paired with an idea from the traditional Vector Quantization (VQ) algorithm, during training. Additionally, the posterior and prior distributions are considered to be categorical.

5.1.1 Discrete Latent Space

The **Discrete Latent Space** is defined as an embedding space $e \in \mathbb{R}^{K \times D}$ where K is the number of categories in the categorical distribution and D is the dimensionality of each embedding vector e_i. As we can see in Figure 5.1.1 the input x (an image) is passed through the encoder to produce $z_e(x)$. The discrete latent variables can then be calcuted through a nearest neighbour lookup on the embedding library (equation 5.1.1). After the quantization process, the now-discretized output of the encoder, z_q is passed on to the decoder as its input. We can calculate z_q as shown in equation 5.1.2. Note that although we represent the latents with a single random variable z, when dealing with images we extract a 2-D latent representation when encoding

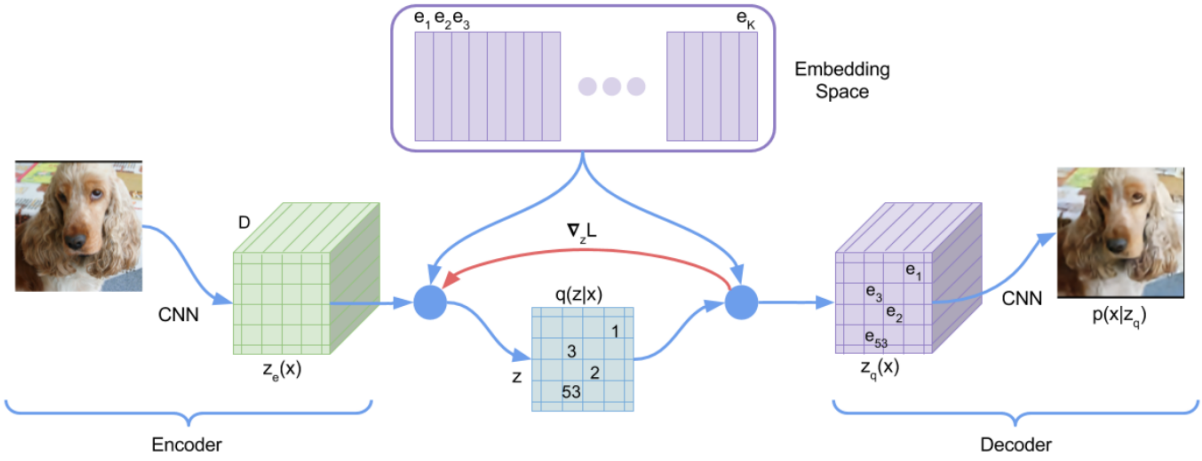


Figure 5.1.1: The Original VQ-VAE architecture [39]

a single image, where each D-dimensional vector in the 2-D representation is quantized according to the embedding library.

$$q(z=k,x)=\begin{cases}1 & \text{for } k=argmin_j||z_e(x)-e_j||_2\\ 0 & \text{otherwise}\end{cases} \tag{5.1.1}$$

$$z_q(x)=e_k, \text{where } k=argmin_j||z_e(x)-e_j||_2 \tag{5.1.2}$$

5.1.2 Encoder and Decoder

The **Encoder** and the **Decoder** are both Convolutional Neural Networks (CNNs). The Encoder comprises of several downsampling blocks and several resnet blocks, that transform the input image tensor $x \in \mathbb{R}^{3\times N\times N}$ into the latent representation $z_e \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}}$ where f is the downsampling factor. This latent vector can then be interpreted as a 2-D grid of D-dimensional vectors, so that each of these vectors can be quantized according to the library of embeddings $e \in \mathbb{R}^{K\times D}$. Symmectically, the decoder takes in the result of the quantization process $z_q \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}}$ and passes it through of sequence of upsampling blocks in order to map it into the image space: $\mathbb{R}^{3\times N\times N}$. The decoder also employs resnet blocks, just as the encoder.

5.1.3 Training

During **Training** we use the loss function specified in Equation 5.1.3. Each one of its three parts is used to optimize a different component of the architecture. The first term, known as the reconstruction loss optimizes the encoder and the decoder. As the name implies it helps the model learn to effectively reconstruct a given image. As the authors of [39] point out, there is no gradient defined for equation 5.1.2, so the propose to just copy the gradients from the decoder to the encoder during backpropagation, in order to promote learning in the encoder as well.

$$\mathcal{L}=\textbf{log}p(x|z_q(x))+||sg[z_e(x)]-e||_2^2+\beta||z_e(x)-sg[e]||_2^2 \tag{5.1.3}$$

Since there is no gradient defined for equation 5.1.2, the library of embedding vectors e_i does not receive any information from the reconstruction loss term. Hence, the second loss term is introduced. This term borrows the idea of the Vector Quantization (VQ) algorithm, to move the embedding vectors e_i towards the encoder outputs $z_e(x)$ according to a squared error loss.

Finally, in order to prevent the volume of the latent space from growing arbitrarily, pushing the embeddings away from each other, the third loss term is introduced. This term essentially discourages encoder outputs

from growing and forces the encoder to "commit" to certain embeddings. This is why it is called commitment loss.

$sg[.]$ in equation 5.1.3 symbolizes the stop gradient operator. It essentially constraints its operand to be a non-updated constant during backpropagation. As it follows from all of the above, the encoder optimizes the first loss terms, the decoder optimizes the first and last loss term and the embedding library only optimizes the middle loss term.

5.1.4 Prior Distribution

The **Prior Distribution** is kept constant during training. After training the VQ-VAE, the authors of the paper fit an autoregressive distribution over z, $p(z)$. For this purpose, they use a PixelCNN [41, 40], over the discrete latent space.

5.2 VQ-GAN

[8] proposes VQ-GAN, a variation of the original VQ-VAE, that employs a slightly different training approach. The authors of the paper attempt to improve the efficiency of the latent encoding, by promoting learning of perceptually rich latent representations in a more compressed space.

VQ-GAN departs from the original VQ-VAE framework in two main ways. Firstly, it replaces the reconstruction loss (first loss term in equation 5.1.3) with a perceptual loss [47]. Secondly, it introduces a patch discriminator and adds auxiliary GAN-style feedback during training. The GAN-style loss function is the traditional one [11]:

$$\mathcal{L}_{GAN}(\{E, G, Z\}, D) = [\mathbf{log} D(x) + \mathbf{log}(1 - D(\hat{x}))] \tag{5.2.1}$$

where E, G, Z and D symbolize the Encoder, Decoder, latent space Codebook and Discriminator respectively.

The complete model is then optimized under the following objective:

$$\mathbf{min}_{E,G,Z}\mathbf{max}_{D}\mathbb{E}_{x\sim p(x)}[L_{VQ-VAE}[E, G, Z] + \lambda L_{GAN}(\{E, G, Z\}, D)] \tag{5.2.2}$$

where L_{VQ-VAE} represents the VQ-VAE's loss function described by equation 5.1.3, where the reconstruction loss is replaced by the perceptual loss as mentioned earlier.

Chapter 6

Transformers as powerful Prior Distributions

As we have already discussed in Chapter 5, in order to use a VQ-VAE as a generative model we need a prior distribution, whether static or learned. For example the authors of [39] used a pixelCNN network to learn an autoregressive prior over the discrete latents. As the prevalence of Transformers models has risen in the previous years, several papers emerged investigating the use of Transformer decoders as priors, paired with VQ-VAEs for conditional image generation. This approach has shown promising results with autoregressive transformers [26, 7, 8] and recently with more efficient, bidirectional, iterative transformers [3, 4].

6.1 DALL-E

Amongst the various works that employ an autoregressive transformer to learn the prior distribution over the image latents, conditioned on text, perhaps **DALL-E** [26] is the most well known. The authors of the paper scaled up both the size of the model (12 billions parameters) and the size of the training set (250 million text-image pairs from the internet) compared to previous approaches. DALL-E managed to produce high-fidelity images and notably achieved high-quality zero-shot image generation on the MS-COCO dataset, even compared to previous models that where trained on the dataset.

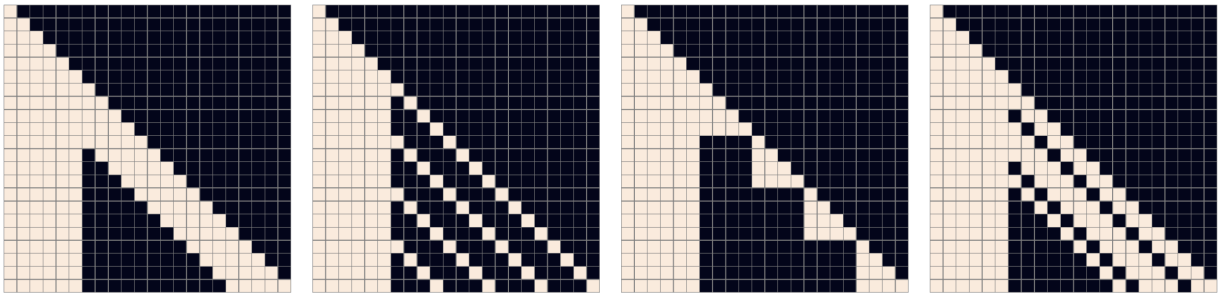
6.1.1 dVAE

The high-level approach that is followed is similar to the one of the original VQ-VAE described in Chapter 5. At first a Discrete VAE (dVAE) is trained. A dVAE is a Variational Autoencoder with a discrete latent space, just like the VQ-VAE. There exist some technical differences between the models that fall out of the scope of this thesis. The dVAE learns to map 256×256 images into a 32×32 grid, whilst training an embedding library of 8192 distinct vectors, that function as the discrete latent space.

6.1.2 BPE-encoding

BPE-encoding is the form of text-encoding used in DALL-E. In order to train a BPE-encoder on a corpus, we first initialize its token vocabulary with all the distinct characters present in the corpus. Then, we repeat the procedure bellow, iteratively, until we reach the desired vocabulary size:

- We find the most frequent pair of tokens in the corpus
- We introduce a new token to the vocabulary that represents this pair of tokens
- We substitute all instances of the pair, in the corpus with the new token



(a) Row attention mask. (b) Column attention mask. (c) Column attention mask with transposed image states. (d) Convolutional attention mask.

Figure 6.1.1: Types of attention masks as described in [26]. The illustration corresponds to a hypothetical version of the transformer with a maximum text length of 6 tokens and image length of 16 tokens (i.e., corresponding to a 4×4 grid). Mask (a) corresponds to row attention in which each image token attends to the previous 5 image tokens. The extent is chosen to be 5, so that the last token being attended to is the one in the same column of the previous row. To obtain better GPU utilization, we transpose the row and column dimensions of the image states when applying column attention, so that we can use mask (c) instead of mask (b). Mask (d) corresponds to a causal convolutional attention pattern with wraparound behavior (similar to the row attention) and a 3×3 kernel. The model proposed by the paper actually uses a mask corresponding to an 11×11 kernel.

6.1.3 Transformer

After the dVAE has been trained, a Transformer Decoder is used to learn the prior distribution over the joined text and image tokens. A text-image pair is encoded as follows: the lowercased text is BPE-encoded [32] into a sequence of at most 256 text tokens, using a vocabulary of 16,384 tokens. The image is mapped into a 32×32 grid of latent tokens, that can the be flattened into a 1-D vector of size 1024. The text and image tokens are then concatenated into a single vector of joint text and image tokens.

The model that is used is a Transformer decoder, that operates autoregressively. For text tokens, causal attention is used. On the other hand, each image token is allowed to attend to all text tokens. Additionally there are three different attention modes used for image-to-image attention (attention between different image tokens). Either a row, column or convolutional attention mask is used. The model employs 64 self-attention layers, each one of them following one of the attention modes mentioned above. All 64 layers have 62 attention heads. The transformer is trained to minimize the cross entropy loss for both the image and text tokens, putting more weight on the loss produced by the image tokens.

As we have already stated, the final model is a 12 billion parameter transformer. The training set used consists of 250 million text-image pairs. It incorporates Conceptual Captions, text-image pairs from Wikipedia and a filtered subset of YFCC100M. The model was reportedly trained on 1024 Nvidia V100 GPUs (16 GB of memory each) .

6.2 MaskGIT

As we have already discussed, DALL-E and other similar approaches (e.g. [8]) successfully used transformers to learn the prior distribution over image latents, conditioning on text. However, these approaches model the image generation task in a completely autoregressive way, i.e. the tokens are produced one by one, with successive passes through the transformer model, from the top left token, to the botton right one. Every token can then attend only to its past tokens during inference. This is obvious from the attention masks in Figure 6.1.1, where all future tokens are masked (The upper right half of the matrix is all masked out). This is an idea that stems from the use of Transformers for text modeling. It is a reasonable idea when thinking about textual data, where words tend to have a sequential ordering and each word mostly depends to the past ones to take meaning.

However, the researchers behind MaskGIT (Masked Generative Image Transformer) [3] argue that this technique is neither optimal, nor efficient for the task of image generation. As they point out, a human artist doesn't necessarily paint an image starting from the top left corner and ending in the bottom right. In fact, he is most likely to start with a small sketch somewhere in the midle of the canvas and progressively refine it, whilst adding more details. Additionally, in contrary to what happens in text, autoregressive modeling scales quadradically for images, due to their inherent matrix-like nature. This deems autoregression quite inefficient, even for relatively small latent image representations, especially since attention mechanisms are already computationally costly. Based on the above observations, MaskGIT is trained to predict all image tokens with a single pass through the transformer model. The tokens are then refined with a small number of iterations, conditioned on the previous generation.

6.2.1 Method

The higher-level approach followed by MaskGIT can be seen in Figure 6.2.1 . It very similar to the two-stage approach employed by VQ-VAE and DALL-E, both of which, we have already discussed in the previous chapters. In the first stage, a discrete latent space, to which we can map images and then reconstruct them from the mapping, is learned. In the second stage, a Bidirectional Transformer (tokens can attend to both directions) is trained to learn the prior distribution of image tokens, conditioned on text captions. In order to train the transformer, the authors of the paper propose a novel training scheme, named *Masked Visual Token Modeling* (MVTM), where a number of randomly selected image tokens are masked and predicted in each training step.

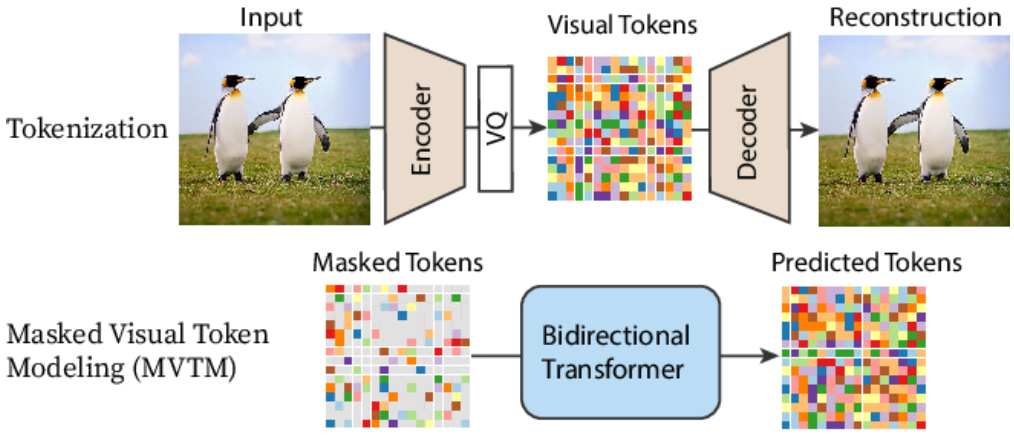


Figure 6.2.1: Overview of the MaskGIT architecture [3]

6.2.2 First Stage

For the first stage, VQ-GAN [8] is employed without any changes. We have already discussed VQ-GAN in Chapter 5. The codebook used in the paper has 1024 tokens and the compression factor is $f = 16$, that is a

256×256 image is mapped into a $\frac{256}{16} \times \frac{256}{16} = 16 \times 16$ grid of latent embeddings.

6.2.3 Second Stage

As we have already hinted, a bidirectional transformer is used on the second stage, through a procedure called MVTM. The transformer that is used has 24 layers, 8 attention heads, 768 embedding dimensions and 3072 hidden dimensions.

MVTM during training

Let $Y = [y_i]_{i=1}^N$ be the latent tokens produced by the VQ-encoder with an image as input and $M = [m_i]_{i=1}^N$ be the binary masks for all tokens. In each training step, we sample a subset of tokens and replace them with a special token ([MASK]). $m_i = 1$ denotes a masked token, while tokens with $m_i = 0$ are not changed. The sampling procedure uses a scheduling function $\gamma(r) \in (0, 1]$. The procedure works as follows:

- A sampling ratio between 0 and 1 is sampled through $\gamma(r)$
- $\lceil \gamma(r) \cdot N \rceil$ tokens are uniformly selected and masked
- Let $Y_{\bar{M}}$ be the resulting token vector after applying mask M over Y
- The model is trained to minimize the negative log-likelihood of the masked tokens:

$$\mathcal{L}_{mask} = -\mathbb{E}[\sum_{\forall i \in [1,N], m_i = 1} \mathbf{log} p(y_i | Y_{\bar{M}})] \tag{6.2.1}$$

Essentially, the probabilities $p(y_i|Y_{\bar{M}}) \in \mathbb{R}^{N \times K}$ are predicted by the transformer. K here denotes the size of the codebook.Then the cross-entropy between them and the ground-truth one-hot token is computed.

Iterative Inference

During Inference a novel parallel decoding method is used, that includes a small number of steps, contrary to the traditional autoregressive decoding in transformers that requires as many steps as the length of the image sequence (e.g. 256 or 1024). A visual comparison of the two methods can be seen in Figure 6.2.2.

Due to the bidirectional nature of the Transformer, the model can theoretically infer all tokens in a single pass. However, the authors of the paper find this challenging, since it is quite different from the training task. Instead they propose a inference method that starts from a "blank canvas", that is all tokens are masked out in $Y_{\bar{M}}^{(0)}$. Then the algorithm in iteration t runs as follows:

- Given the masked tokens $Y_{\bar{M}}^{(t)}$ at the current iteration, the probabilities $p^{(t)} \in \mathbb{R}^{N \times K}$ for all masked tokens are predicted using the transformer.
- At each masked location i, a token $y_i^{(t)}$ is sampled based on the prediction probabilities $p_i^{(t)} \in \mathbb{R}^K$ (treating $p_i^{(t)}$ as a multinomial distribution). After sampling, the probability $p_i^{(t)}$ of the token that was sampled is now used as a "confidence" score, showing how "confident" the model is about this prediction. For the unmasked positions, the confidence scores are set to 1.0.
- The number of tokens to (re)mask is computed as $n = \lceil \gamma(\frac{1}{T})N \rceil$, where γ is the mask scheduling function, N is the total number of tokens and T is the total number of iterations.
- We obtain the masked tokens for the next iteration $Y_{\bar{M}}^{(t+1)}$, by applying the new mask $M^{(t+1)}$ that is derived by the following formula:

$$m_i^{(t+1)} = \begin{cases} 1 & \text{if } c_i < sorted_j(c_j)[n] \\ 0 & \text{otherwise} \end{cases} \tag{6.2.2}$$

  where c_i denotes the confidence score for the i-ith token.

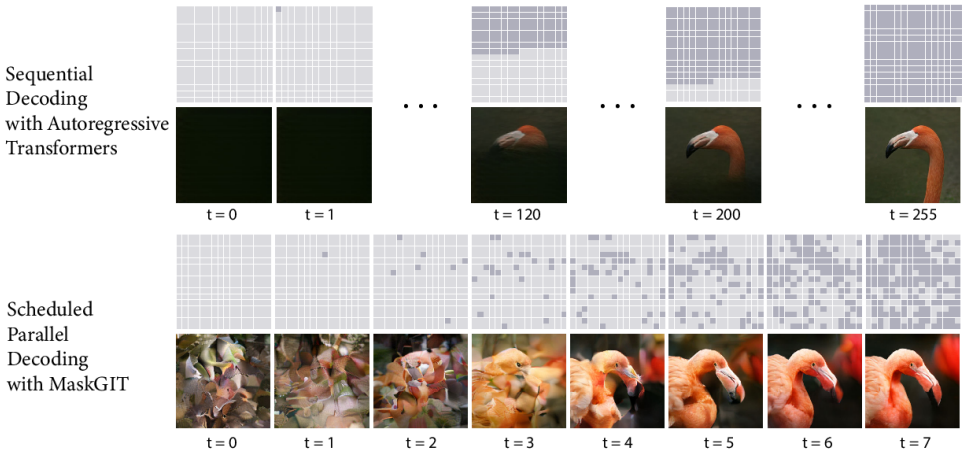


Figure 6.2.2: Autoregressive Inference vs Parallel Iterative Decoding used in MaskGIT [3]

In short the model generates an image in T iterations. In each iteration, it keeps the predicted tokens that it is most confident about and remasks the rest of them, so as to predict them in future iterations. The masking ratio is descreased till all tokens are predicted within T iterations.

Masking Design

The authors of the paper report that the way they model masking has a significant impact on the quality of the generated images. As we saw, masking is scheduled by a function $\gamma(.)$, that is used in both training and inference. During training it takes a random ration $r \in (0, 1]$ as input, while at Inference Time, it takes $0/T, 1/T, ..., (T-1)/T$ depending on the timestep we are at.

γ needs to have the following properties:

- It should be a continuous function $\in [0, 1]$, for inputs $r \in [0, 1]$.
- It needs to be monotonically decreasing with respect to r, with the property $\gamma(0) \to 1$ and $\gamma(1) \to 0$. This is essential for the inference algorithm to converge.

In the paper, three families of functions are tested: *Linear*, *Concave* (including cosine, square and exponential) and *Convex* (including square root and logarithmic) functions. Some examples of such functions can be seen in Figure 6.2.3. The authors report that the **Cosine** function performs the best in all of their experiments.

Number of Iterations

Concerning the number of iterations during inference, the researchers report that the optimal number falls between 8 and 12. Intuitively one might think that more iterations would yield a finer result. However, it is hypothesized that such a sweet spot exists because too many iterations would statistically eliminate less confident tokens and lead the generation process to collapse to a subset of very likely tokens, hindering diversity.

6.2.4 Token-Critic

Token-Critic [16] is proposed as a method to improve the token sampling procedure of non-autoregressive Transformer models, like MaskGIT. As we have already discussed, during inference, MaskGIT follows an iterative parallel decoding scheme where subsets of the full token set are predicted over a number of steps. At each step, the most confident tokens are kept. In order to make this decision, we view the logits, based on which we sampled the tokens, as their confidence scores. Instead of reusing logits as confidence scores, Token-Critic can be trained as an auxiliary model, whose purpose is to estimate the confidence scores for the predicted tokens.

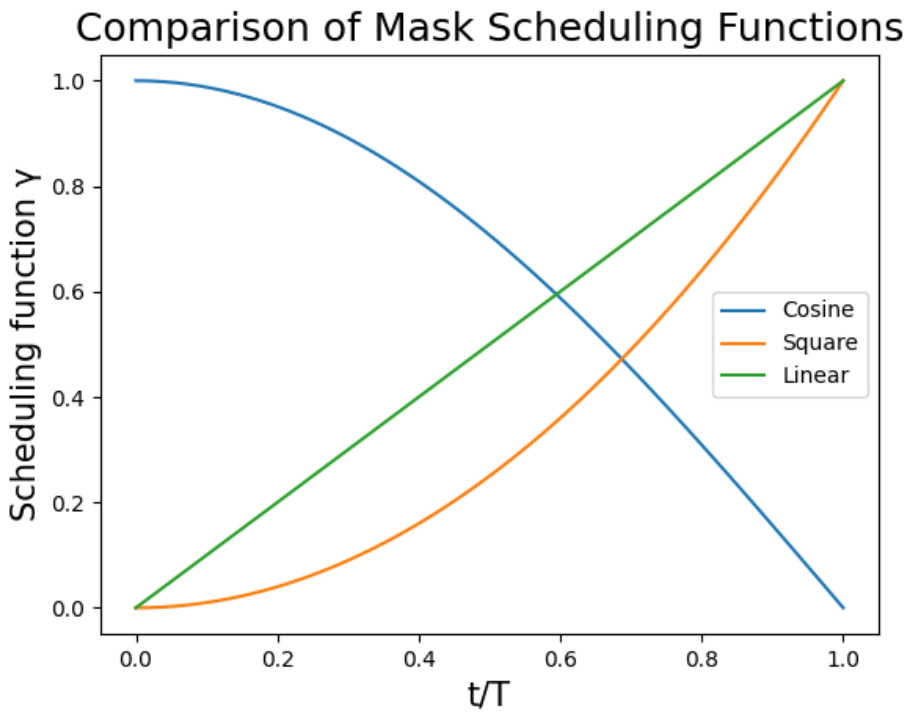


Figure 6.2.3: Comparison of Mask Scheduling Functions

Training

Token-Critic is a transformer, typically of smaller size than MaskGIT's generative transformer. First, we assume that we already have a trained MaskGIT. At training time, $Y = [y_i]_{i=1}^N$ are the latent tokens produced by the VQ-encoder, from an image. $M = [m_i]_{i=1}^N$ are the binary masks for all tokens. $Y_{\bar{M}}$ be the resulting token vector after applying mask M over Y. Given the masked tokens, we sample $\hat{Y}$ from $p(y_i|Y_{\bar{M}})$ (this is the distribution parameterized by MaskGIT). Then, we form $\tilde{Y} = \hat{Y} \odot (1 - M) + Y \odot M$. $\tilde{Y}$ is essentially equivalent to substituting each one of the [MASK] tokens in $Y_{\bar{M}}$ with the corresponding token predicted by MaskGIT. Token-Critic is trained to minimize:

$$\mathcal{L} = \mathbb{E}[\sum_{j=1}^{N} BCE(m_j, p_\phi(m_j|\tilde{Y}, c))] \tag{6.2.3}$$

Where $p_\phi(.)$ is parameterized by the Token-Critic. That is, Token-Critic is trained to predict the binary Mask, by conditioning on $\tilde{Y}$ and c, where c is a condition (e.g. text condition).

Inference

For inference, the iterative design from MaskGIT is used, with a small modification. At first, we start with a "blank canvas" $Y_{\bar{M}_1}^{(0)}$ (all tokens masked). At step t, we sample $Y^{(t)}$ using MaskGIT, i.e.:

$$Y^{(t)} \sim p_\theta(Y^{(t)}|Y_{\bar{M}_t}^{(t-1)}, c) \tag{6.2.4}$$

where $p_\theta(.)$ is the distribution learned by MaskGIT.

Now, instead of using the logits that we used to sample $Y^{(t)}$ (logits that where produced by MaskGIT) as confidence scores, based on which we will remask the least confident tokens, we employ the Token-Critic. We sample $M_{t+1} \sim p_\phi(M_{t+1}|Y^{(t)}, c)$ ($p_\phi(.)$ is parameterized by the Token-Critic). $M_{t+1} \in \mathbb{R}^N$ contains N values, all of them between 0 and 1, corresponding to the confidence scores for each one of the predicted tokens.

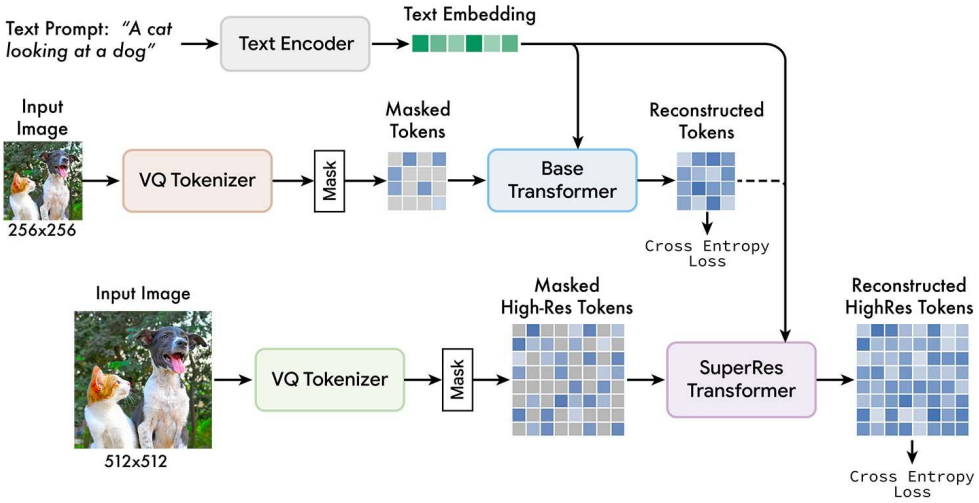


Figure 6.3.1: Overview of Muse's architecture: The T5-XXL pre-trained text encoder, the base model and the super-resolution model are depicted on the three rows. The text encoder generates a text embedding that is used for cross-attention with image tokens for both base and super-res Transformer layers. The base model uses a VQ Tokenizer that is pre-trained on lower resolution (256×256) images and generates a 16×16 latent space of tokens. This sequence is masked at a variable rate per sample and then the cross-entropy loss learns to predict the masked image tokens. Once the base model is trained, the reconstructed lower-resolution tokens and text tokens are passed into the super-res model that then learns to predict masked tokens at a higher resolution. [4]

Since Token-Critic is a transformer, it uses attention to take into account correlation between tokens when predicting the confidence scores, which improves the quality of this decision, compared to the independent sampling originally used by MaskGIT.

6.3 Muse

More recently (January 2023), another paper [4] was published by Google researchers, building on the same philosophy as MaskGIT, whilst proposing several improvements. The new model, named Muse, employs a three-stage approach instead of the two-stage approaches we have discussed in the previous chapters, including image tokenization (VQ-GAN) a base-resolution Transformer and a super-resolution Transformer. The Transformers are Bidirectional and follow the same random masking training scheme and iterative, parallel decoding tactic that was used in MaskGIT. The model also leverages text embeddings from a pretrained LLM (Large Language Model), instead of learning text representations from scratch.

6.3.1 Model

An overview of the Model can be seen in Figure 6.3.1. In the following section we analyze the most important components of the architecture.

Pre-Trained Text Encoder

The encoded text input that is fed into the model is extracted using a T5-XXL [24] model, which belongs to the LLM category. The embeddings from the LLM are supposed to encode rich information about objects, actions, visual properties, spatial relationships and other properties. Muse is then expected to map this information into the generated images.

In terms of method, an input text caption is passed through a frozen T5-XXL encoder resulting in sequence of 4096 dimensional language embeddings, which are then projected to the hidden size of the transformer and used for cross-attention.

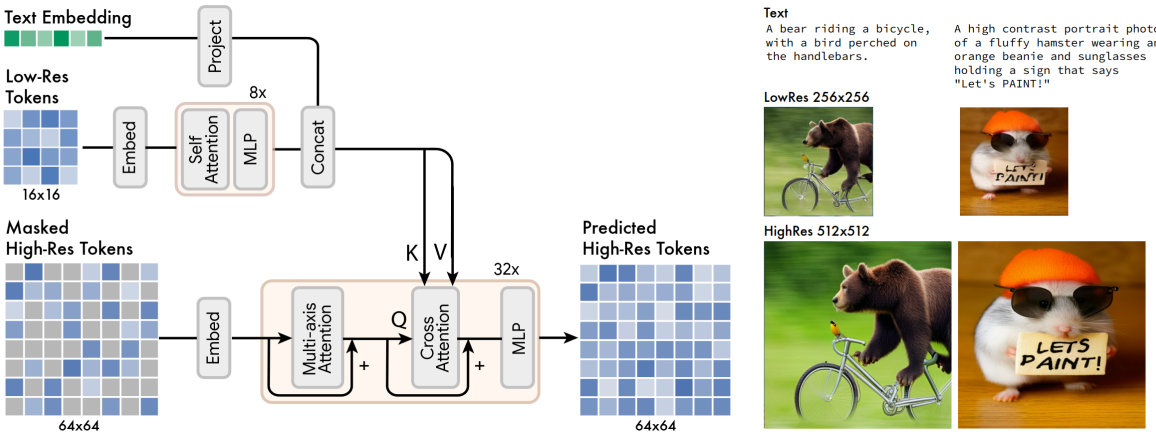


Figure 6.3.2: Muse's super-resolution model [4]

Image Tokenization: VQ-GAN

For image tokenization, VQ-GAN is used here, as in the case of MaskGIT. In fact two separate VQ-GAN models are trained. The first one is trained on 256×256 images with a downsampling factor f=16, resulting in a 16×16 latent representation. This one is used for the base model. The second VQ-GAN, that is used to train the super-res model, operates on 512×512 images, with a downsampling factor f=8, resulting in a 64×64 latent representation. Both of them have a library of 8192 embeddings.

Base Model

As we have already mentioned, the base model is a Bidirectional Transformer. Essentially, it operates in an identical way to MaskGIT. It takes as input the T5 text embeddings and the image tokens. The Base Model uses the image tokens outputed by the first VQ-GAN model, that encodes 256×256 images into a 16×16 grid of latents. During training a random fraction of the image tokens is masked and predicted. At inference, Iterative Parallel Decoding is employed, as in MaskGIT (6.2.3).

Super-Resolution Model

An overview of the Super-Resolution Transformer can be seen in Figure 6.3.2. This part of the model is trained, with a pre-trained frozen base model. The Super-Resolution model takes in as input the t5 text embeddings, the Low-Resolution image tokens, i.e. the 16×16 grid predicted from the base model. These tokens are passed through a simple, short Transformer Network, only employing Self-Attention and MLP layers. The Super-Res Transformer is a Bidirectional Transformer, just like the base model, that operates on the high resolution latent space of the second VQ-GAN model, i.e the 64×64 grid produced by 512×512 images. Apart from the different representation spaces, the only other difference between the base and the Super-Res model, is that the second one cross-attends to the low resolution tokens produced by the base model in addition to the text embeddings.

Intuitively, the Super-Res model learns a mapping between the Low-Res representation vectors and the High-Res ones. The researchers reported finding that directly predicting high resolution images (512×512) lead the model to focus on fine details and ignore large scale semantics. Therefore, they went for this gradual approach to capture both high-level and low-level features.

Inference: Iterative Parallel Decoding

Decoding follows the tactic that was introduced in MaskGIT. A cosine Masking Schedule is employed, to decide the fraction of tokens that are to be predicted in each step. Then, at each timestep the most "confident" tokens that fill up this fraction are kept and the rest of the tokens are re-masked and left to be predicted in the next steps. The Base model predicts its output over 24 steps vs 8 steps for the Super-Res model, for a total of 32 timesteps.

Classifier Free Guidance

The model reportedly employs a technique called Classifier Free Guidance [12]. At training time, a fraction (10%) of samples is chosen randomly and the text condition is completely dropped for them (thus attention reduces to image token self-attention). At inference time, two seperate logits are computed for each image token: a conditional logit ℓ_c (using text condition) and an unconditional logit ℓ_u (not using text condition). the fianl logits are computed as follows:

$$\ell = (1 + t)\ell_c - t\ell_u \tag{6.3.1}$$

where t is the guidance scale. Classifier Free Guidance intuitively trades of diversity for fidelity. Specifically, a higher value of t promotes higher fidelity, but also lower diversity, by increasing the focus of the model on the conditional logits.

The mechanism is also used to enable Negative Prompting. Specifically, by replacing the unconditional logits with logits conditioned on a Negative Prompt, Equation 6.3.1 can be used to push the final logits towards features encouraged by the positive prompt (ℓ_c) and away from the features encouraged by the negative prompt (ℓ_u), thus effectively disabling unwanted features.

Chapter 7

Caption Augmentation using LLMs

The outstanding capabilities of LLMs have been previously leveraged to perform text augmentation in the context of various tasks [38, 43, 6, 44, 9]. [9] proposes a method for augmenting captions of text-image pairs that are used to train a CLIP[23] model. At first, alternative captions are generated for a small number of text-image pairs, through various methods, including human annotation and chatbots. Original and generated captions are paired to form meta-input-output pairs. Subsequently, LLaMA[37] is used to produce alternative captions for all samples in the training data. The meta-input-output pairs are used as context for the LLM to better understand the task.

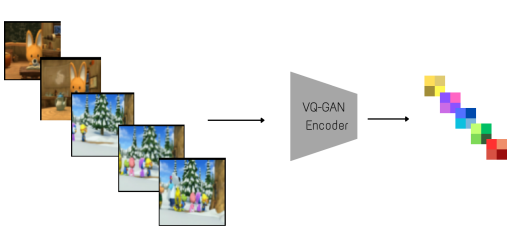
Chapter 8

Masked Generative Story Transformer

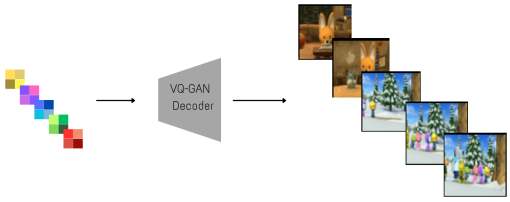
In this Chapter we introduce our approach for the task of Story Visualization, analyzing each main component independently.

8.1 Image Tokenization

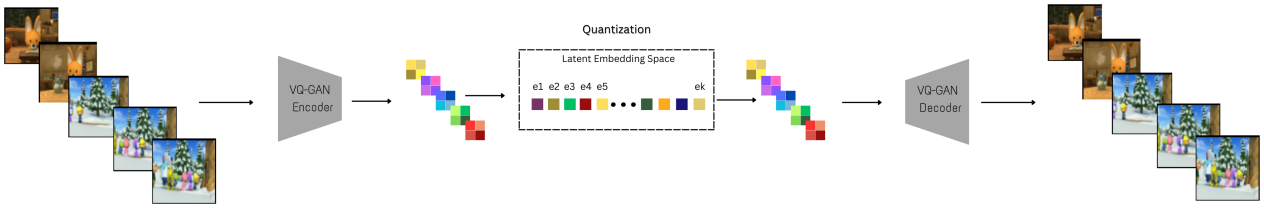
8.1.1 VQ-GAN



(a) The images of a story are tokenized by the VQ-GAN encoder



(b) The latent image tokens can be decoded into the image space using the VQ-GAN decoder



(c) VQ-GAN's function

Figure 8.1.1: VQ-GAN encoder and decoder

For Image tokenization we leverage VQ-GAN [8]. Given the images of a story: $X = \{X_1, X_2, ..., X_n\}$, we pass them through the VQ-GAN encoder to produce the corresponding discrete latent tokens: $Z = \{Z_1, Z_2, ..., Z_n\}$, as show in Figure 8.1.1a. A Transformer is trained to predict the image tokens based on text (see Section 8.3). After the image tokens have been predicted, we can decode them back into the image space, using the VQ-GAN decoder (Figure 8.1.1b).

8.2 Text Encoding

For Text-Encoding we experiment with two alternative methods: training text-embeddings from scratch or using text embeddings extracted from an LLM. Regardless of the method, each input caption in a story is

encoded into sequence of embeddings. We denote the encoded texts of a story as $T = \{T_1, T_2, ..., T_n\}$.

8.2.1 Custom Text Embeddings

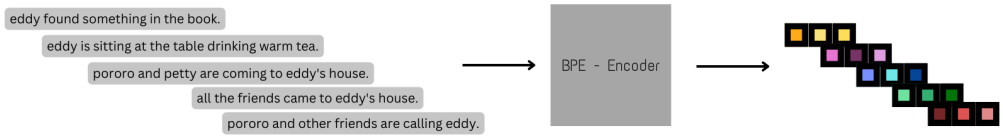


Figure 8.2.1: The sentences of a story are BPE-encoded to obtain text tokens

When training text embeddings, we use a BPE-encoder that maps each text-description into a sequence of image tokens (Figure 8.2.1). For the BPE-encoder we use a vocabulary of size 2500.

8.2.2 Using an LLM

Instead of training our own text embeddings, we can use the embeddings from a pretrained LLM. Inspired by MUSE [4], we experiment with T5-XXL [24]. To produce embeddings from a text caption, we pass the caption through the LLM and extract the embeddings from the final hidden layer. Then, we use them as the text representations that serve as input to our transformer.

8.3 Transformer Priors

8.3.1 Input

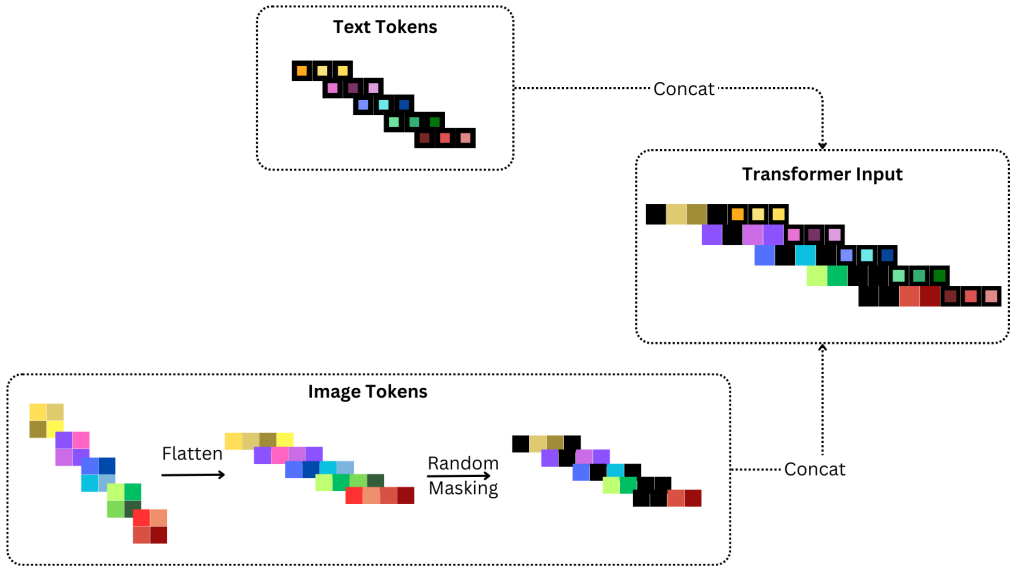


Figure 8.3.1: The input of the Transformer is the concatenation of the image and text tokens for each one of the five image/sentence pairs in the story. The Image Tokens are flattened and randomly masked before being concatenated with the text tokens.

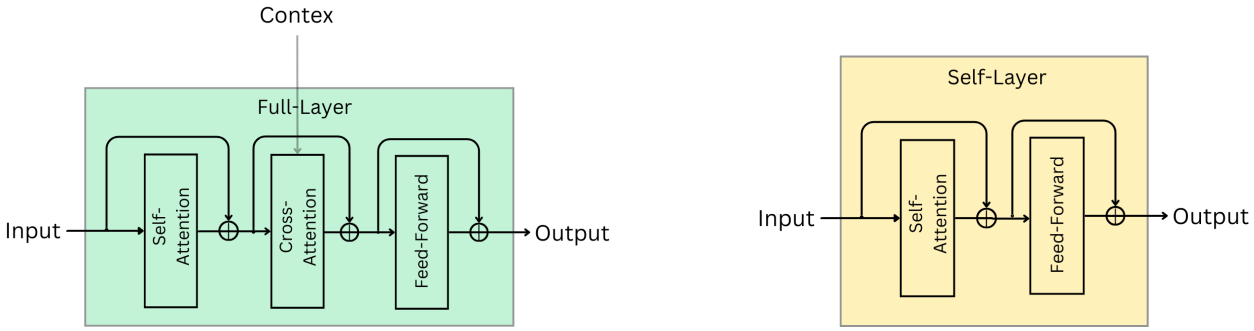
Given the image tokens $Z = \{Z_1, Z_2, ..., Z_n\}$ ($Z_i \in \mathbb{R}^{m \times m \times d}$) and text tokens $T = \{T_1, T_2, ..., T_n\}$ ($T_i \in \mathbb{R}^{l \times d}$), for a story, we form the transformer's input as shown in Figure 8.3.1. The image tokens $Z \in \mathbb{R}^{n \times m \times m \times d}$ are flattened into a sequence $Z' \in \mathbb{R}^{n \times (m \cdot m) \times d}$. Then they are randomly masked, as in *MaskGIT* to obtain $\bar{Z} \in \mathbb{R}^{n \times (m \cdot m) \times d}$. Finally, each image representation is concatenated with the text representation of the same index to form $Input_i = (\bar{Z}_i; T_i)$. The Transformer's input can be written as :

$$Input = \{Input_1, ..., Input_n\} \in \mathbb{R}^{n \times (m \cdot m + l) \times d} \tag{8.3.1}$$

where l is the length of a captions text representation, $m \times m$ is the resolution of the images' latent representations, n is the number of images in a story and d is the Transformer's hidden dimension.

8.3.2 Types of Transformer Layers

Below, we describe the different types of Layers we use in out transformer models.



(a) Full-Layer (b) Self-Layer

Figure 8.3.2: Basic Transformer Layers

Full-Layer

The *Full-Layer* 8.3.2a is the traditional Transformer Decoder Layer, consisting of three sub-layers: Self-Attention, Cross-Attention and Feed-Forward. Given an input $I \in \mathbb{R}^{n \times a \times d}$ and a context $C \in \mathbb{R}^{n \times c \times d}$, the output a Self-Attention sub-layer and a Cross-Attention sub-layer can be calculated, respectively as follows:

$$\begin{aligned} Self - Attention(I) &= MultiHead_1(Q = I, K = I, V = I) \\ Cross - Attention(I, C) &= MultiHead_2(Q = I, K = C, V = C) \end{aligned} \tag{8.3.2}$$

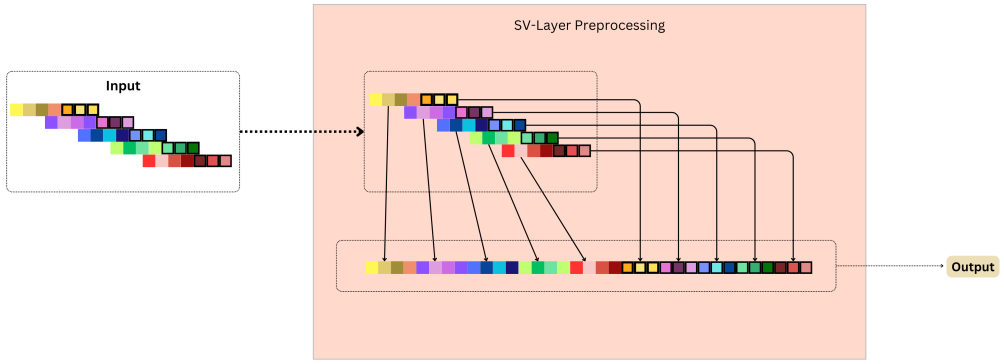
The Feed-Forward sub-layer consists of a Linear Layer, followed by an activation, followed by Layer Normalization. As it it shown in the Figure, there is a residual connection around each sub-layer.

Self-Layer

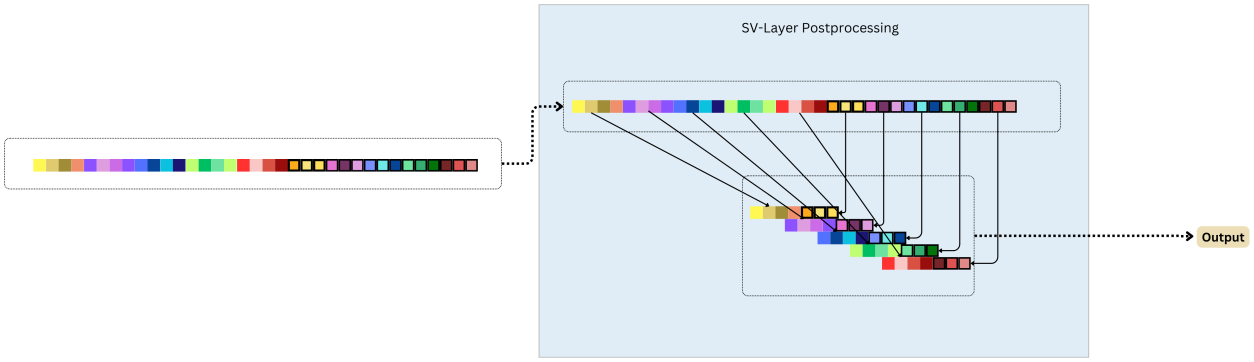
As one can see in Figure 8.3.2b, the Self-Layer is identical to the Full-Layer, except for the fact that it omits the Cross-Attention sub-layer. It, therefore does not utilize any context.

SV-Layer

The SV-Layer (Story Visualization Layer) is shown in Figure 8.3.4. It comprises of a Self-Layer, preceded by a preprocessing sub-layer and followed by a postprocessing sub-layer.



(a) SV-Layer Preprocessing



(b) SV-Layer Postprocessing

Figure 8.3.3: Data Processing for the SV-Layer

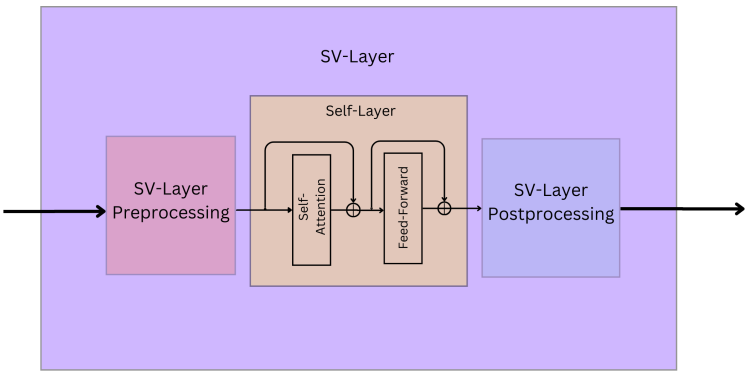


Figure 8.3.4: SV-Layer

The **preprocessing** sub-layer takes in an n-sized batch of tokens $\in \mathbb{R}^{n\times(m\cdot m+l)\times d}$, that corresponds to the n image-caption pairs in a story. It reshapes its input into a representation $\in \mathbb{R}^{1\times(n\cdot(m\cdot m)+n\cdot l)\times d}$ by stacking the image tokens for all images in the story, one next to the other and concatenating them with the sequence of all text tokens in the story stacked in the same way (Figure 8.3.3a).

The **postprocessing** sub-layer has the exact opposite function. It takes in a representation $\in \mathbb{R}^{1\times(n\cdot(m\cdot m)+n\cdot l)\times d}$. It reshapes it into an output $\in \mathbb{R}^{n\times(m\cdot m+l)\times d}$, where each one of the $((m\cdot m+l)\times d)$-sized sequences in the n-batch corresponds to the image tokens of an image, concatenated with the text tokens of its caption (Figure 8.3.3b).

At a conceptual level, the preprocessing sub-layer brings the story representation from an image-by-image format into a story-level format, where the whole story is viewed as a continuous sequence of tokens that correspond to all images and captions. This way, in the Self-Layer that follows, tokens from any one of the

n positions in the story can attend to tokens from any other position, and integrate relevant information. Subsequently, we bring the tokens back to their image-by-image format with the postprocessing sub-layer.

8.3.3 Proposed Transformer Models

MaskGST

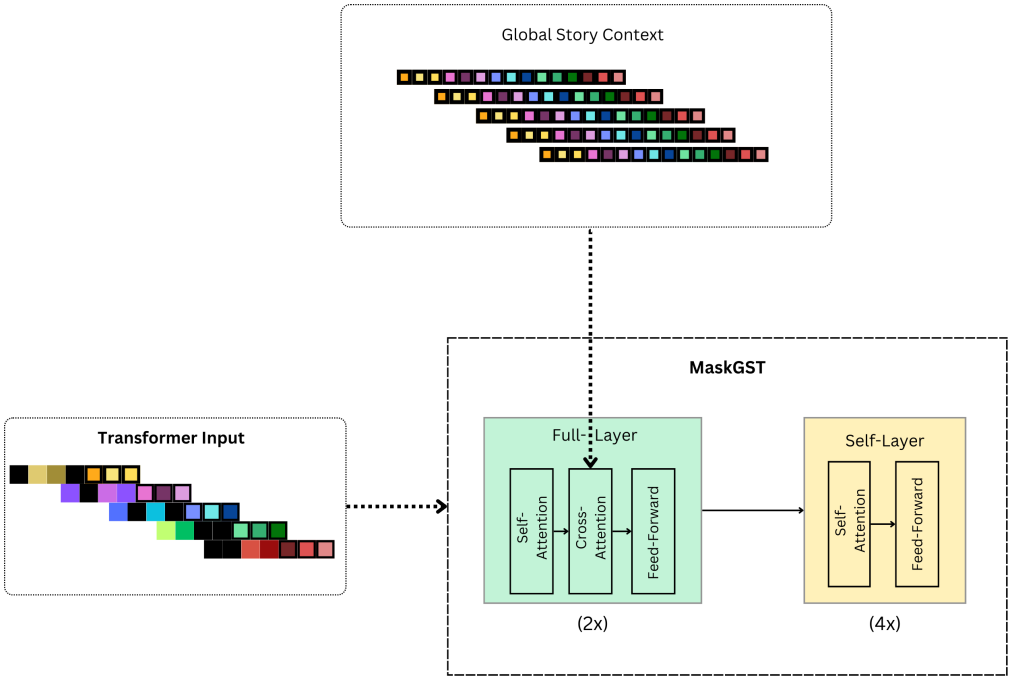


Figure 8.3.5: MaskGST model

We name our proposed Transformer Architecture MaskGST (Masked Generative Story Transformer) to highlight its heavy influence from MaskGIT. MaskGST consists of two Full-Layers, followed by several Self-Layers. Figure 8.3.5 shows a version of the model that employs 4 Self-Layers, for a total of 6 Layers.

The context that is utilized in the cross-attention sub-layers is composed of all the text descriptions in a story. That is, to predict the image tokens for each image in the story, we perform cross-attention with all of the text descriptions in the story. This way, we allow the model to adopt useful, relevant information from previous and following timesteps in the story. For example a character could be referenced by his name (e.g. *Pororo*) in one caption and then be referenced using a pronoun (e.g. *he*) in the next caption. By employing cross-attention with all other captions in the story, we allow the model to learn how to infer the name of the character (or other relevant information) in such cases.

MaskGST-SV

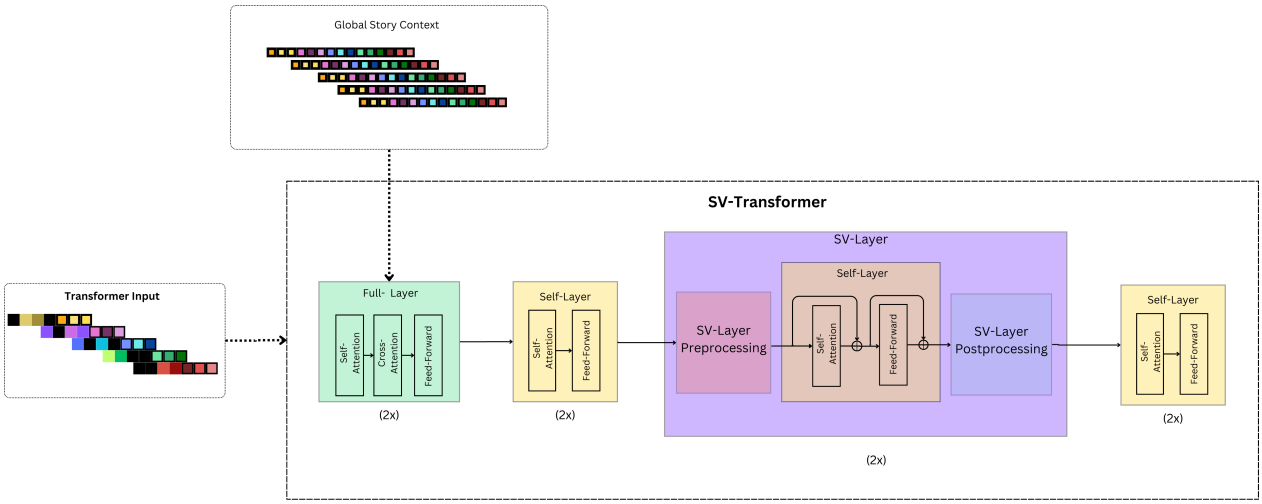


Figure 8.3.6: MaskGST-SV

MaskGST-SV (MaskGST Story Visualization) includes two Full-Layers in the beggining of the generative process. The first two layers can be optionally followed by several Self-Layers. Then, we have two SV-Layers, that can be optionally followed by several Self-Layers, as well. For example, Figure 8.3.6 shows a MaskGST-SV model that stacks two Self-Layers between the Full-Layers and the SV-Layers and two additional Self-Layers after the SV-Layers, for a total of 8 layers. We conduct experiments with several different configurations.

Training

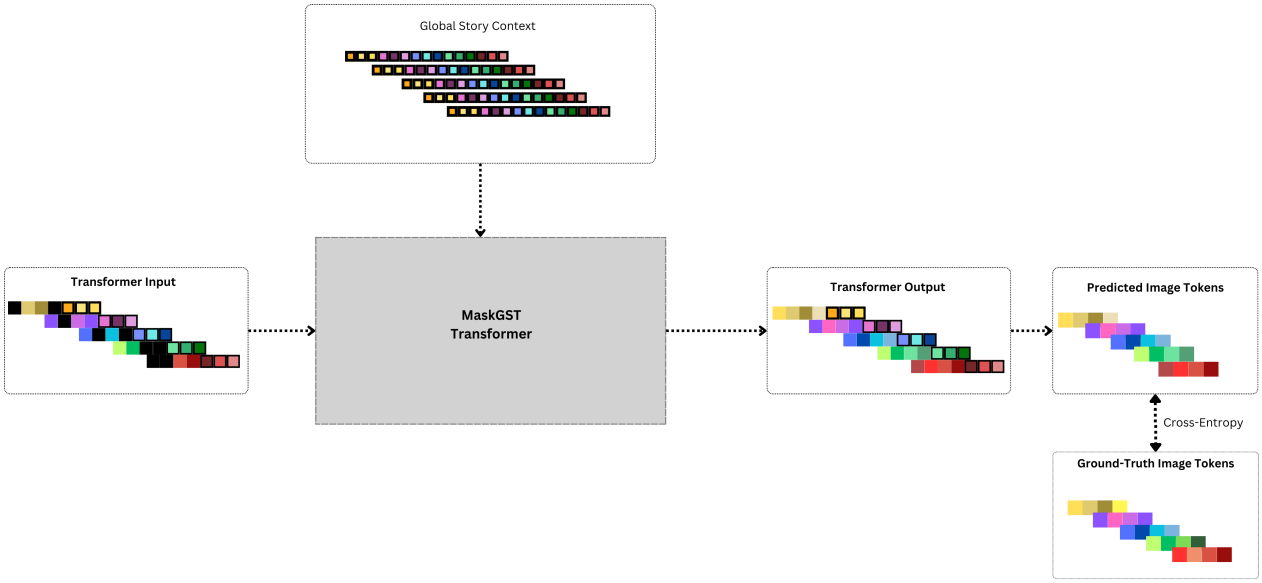


Figure 8.3.7: Overview of MaskGST's training procedure

All of the proposed models transform their input: $Input = \{Input_1, ..., Input_n\} \in \mathbb{R}^{n \times (m \cdot m + l) \times d}$ into an output of the shame shape:

$$Output = \{Output_1, ..., Output_n\} \in \mathbb{R}^{n \times (m \cdot m + l) \times d} \tag{8.3.3}$$

For each item in the n-batch, we drop the l tokens that correspond to the text description and keep the $m \cdot m$ image tokens. Thus, we obtain:

$$\overline{Output} = \{\overline{Output_1}, ..., \overline{Output_n}\} \in \mathbb{R}^{n \times (m \cdot m) \times d} \tag{8.3.4}$$

The output then undergoes a linear transformation to obtain logits:

$$y = \mathbf{Lin}(\overline{Output}) \in \mathbb{R}^{n \times (m \cdot m) \times K} \tag{8.3.5}$$

where K is the number of embeddings in the embedding library of the VQ-GAN. Finally, the logits can be used to train the model against the ground-truth image tokens that are obtained through the VQ-GAN. For training, we use the masking scheme proposed in MaskGIT, that is presented in Section 6.2.3.

If we are working with a VQ-GAN, that uses more than one latent space, we employ as many linear transformations as the latent spaces and use each one of them to predict the tokens of one latent space.

Inference

As in training, for inference we also adopt the parallel decoding procedure proposed in MaskGIT, that is detailed in Section 6.2.3

8.3.4 Character Guidance

We propose a new technique to improve the generation of characters in the stories. We add to our model, a library of $2 \cdot C_n$ extra Character Embeddings, where C_n is the number of main characters present in the dataset (e.g. for Pororo-SV $C_n = 9$). For each character, we have a positive embedding (the character is referenced in the caption) and a negative one (the character is not referenced in the caption). When using this technique, we concatenate C_n embeddings to the input of the transformer, one for each character. Positive embeddings are used for the characters that are present in the current text description and negative ones for the rest of the characters. In that case the input of the transformer becomes:

$$Input = \{Input_1, ..., Input_n\} \in \mathbb{R}^{n \times (m \cdot m + l + C_n) \times d} \tag{8.3.6}$$

where: $Input_i = (\bar{Z}_i; T_i; C_i)$ and $C_i = \{pos_{char}\}_{\text{char} \in T_i} \bigcup \{neg_{char}\}_{\text{char} \notin T_i}$. pos_{char} represents the positive embedding for character $char$, whereas neg_{char} represents the negative embedding for the character. $\{pos_{char}\}_{\text{char} \in T_i}$ is the set of positive embeddings for the characters that are present in the current description (T_i) and $\{neg_{char}\}_{\text{char} \notin T_i}$ is the set of negative embeddings for the characters that are **not** present in the description.

Training

In order to reinforce the model's focus on these Character Embeddings, we completely drop the text descriptions for a percentage of training samples in each batch and only keep the Character Embeddings. Other than that, the training process remains intact.

Inference

During inference, we compute two sets of logits when generating an image. The first set of logits (ℓ_{tc}) is computed conditioning the generative process on text descriptions. The second set of logits (ℓ_{char}) is computed using only Character Embeddings and completely dropping the captions. As in training, we use positive embeddings for characters that are present in the current caption and negative ones for the remaining characters. The final logits (ℓ) are computed as a convex combination of the text-conditional logits and the character logits:

$$\ell = (1 - f)\ell_{tc} + f\ell_{char}, \;\; f \in [0, 1) \tag{8.3.7}$$

This combination is computed at every step of the iterative parallel decoding process.

The intuition behind this approach is that the text-conditional logits encode the specific information regarding the generation of an image from its descriptions, whereas the character logits encapsulate information solely regarding the presence of main characters in the image. By combining the two sets of logits we attempt to produce images that remain faithful to the corresponding descriptions, whilst being specifically biased towards the generation of characters that need to be present.

Negative Prompting

In order to further reinforce the presence of the correct subset of the main characters in each image, we employ negative prompting, during inference. Specifically, except for the two sets of logits (ℓ_{tc} and ℓ_{char}), we introduced earlier, we compute a third set of logits $\ell_{\overline{char}}$. To compute $\ell_{\overline{char}}$ we completely drop text descriptions from the transformer's input, as we do when computing ℓ_{char}. However, instead of using positive embeddings for characters present in the description and negative ones for absent characters, we do the opposite. Negative embeddings are used for characters present in the description and positive embeddings for characters absent in the description. That is, the character embeddings in the input are: $C_i = \{neg_{char}\}_{\text{char}\in T_i} \bigcup \{pos_{char}\}_{\text{char}\notin T_i}$. In a sense $\ell_{\overline{char}}$ is computed using the "*complement*" of the input that is used to generate ℓ_{char}. The final logits are now computed as follows:

$$\ell = (1-f)\ell_{tc} + 2f\ell_{char} - f\ell_{\overline{char}},\ f \in [0,1) \tag{8.3.8}$$

The intuition behind this idea is that if the addition of ℓ_{char} pushes the logits towards generating the characters present in the description, the subtraction of $\ell_{\overline{char}}$ actively pushes the logits away from the generation of characters that are **not** in the current description.

The whole idea behind Character Guidance is akin to the way Classifier Free Guidance [12] is used in MUSE. However, in our approach we introduce Character Embeddings and use the technique to simultaneously focus on the text condition and on the the characters to be generated, instead of trading off diversity for fidelity.

8.4 Caption Set Augmentation

In order to shield our model against overfitting we experiment with augmentation of the text descriptions (image captions). To that end, we employ an LLM. The LLM is instructed to generate alternative captions, given the original one for each story.

We use ChatGPT 3.5 through the API provided by OpenAI to augment our training captions. For each story (set of 5 captions) we give ChatGPT a description of its role, as a caption-augmenting assistant, along with information about the characters in the dataset. Subsequently, we provide it with the five captions of the story, that it needs to rephrase. This technique is closely related to the one adopted in [9]. However in [9] examples of input-output caption pairs are provided as context, whereas we choose to provide domain specific knowledge about the dataset as context. Additionally, we use the alternative captions while training a generative model, where caption accuracy is of grater importance than contrastive language-image training, which is the task in [9].

The role message is the following (Pororo Dataset):

```
 "You are helping me generate more descriptive captions from the given text descriptions.
I want short, simple, visual captions to train a text to image model.  Five consecutive
descriptions form a coherent story.  The main characters in the descriptions are the
following:  Pororo is a Penguin.  Loopy is a pink Beaver.  Crong is a green Dinosaur.
Eddy is a brown fox.  Poby is a polar Bear.  Petty is a blue female Penguin.  Tongtong is
an Orange Dragon.  Rody is a yellow Robot.  Harry is a pink Bird."
```

The original story captions are given in the following form:

```
1.  {caption 1}
2.  {caption 2}
3.  {caption 3}
```

```
4.  {caption 4}
5.  {caption 5}
```

ChatGPT returns the alternative captions:

```
1.  {alt.  caption 1}
2.  {alt.  caption 2}
3.  {alt.  caption 3}
4.  {alt.  caption 4}
5.  {alt.  caption 5}
```

This technique is closely related to the one emloyed in [9]. However in our case, domain specific knowledge about the dateset is given as context, instead of providing ChatGPT with examples of input-output caption pairs. Furthermore, in [9] the captions are used to train a contrastive language-image model[23], while we use the captions to train a generative model, where caption accuracy is of greater importance.

During training, we randomly pick either the original or the alternative caption for each image, on every epoch. This way, we essentially provide different text descriptions for the same image on different epochs. On the one hand, we expect this to help our model focus more on the word embeddings that are more important and more relevant to the visual concepts in the image (e.g. the names of the characters). Additionally, we hypothesise that this technique could improve the models understanding of the variance of language and help it avoid overfitting to the expressions used in the original training corpus.

8.5 Character-Attentive Token-Critic

Based on the original idea of the Token-Critic we experiment with a Character-Attentive Token-Critic. Its function is identical to the one described in Section 6.2.4. For our task, we choose to condition the predictions of the token-critic using Character Embeddings. Specifically, the Token-Critic has C_n embeddings, one for each main character(C_n is the number of characters in the dataset). We train an auxiliary Transformer (Token-Critic) that parameterizes $p_\phi(.)$ under:

$$\mathcal{L} = \mathbb{E}[\sum_{j=1}^{N} BCE(m_j, p_\phi(m_j|\tilde{Y}, c))] \tag{8.5.1}$$

where $[m_j]_{j=1}^N$ is the binary mask, used in the MaskGIT training process, $\tilde{Y} = \hat{Y} \odot (1 - M) + Y \odot M$ and c (the condition) is formed by concatenating the Character Embeddings of the characters that are present in the current caption. Conditioning is implemented through Cross-Attention.

During inference, the Character-Attentive Token Critic is used to predict confidence scores for the generated image tokens, at each inference step, whilst attending to Character Embeddings.

8.6 Latent Super-Resolution Model

Inspired by MUSE, we experiment with the idea of using a Super-Resolution Transformer Model, with a higher latent resolution, that conditions on outputs of a Base Transformer model, with a more compact latent space. For this idea to work, we assume that we have two VQ-GAN models, operating at two different resolutions: $m_1 \times m_1$ and $m_2 \times m_2$ $(m_1 < m_2)$.

8.6.1 Base Transformer

The Base Transformer model is MaskGST, like the one described in Section 8.3.3 and trained as we describe in that Section, using MaskGIT's formula. This Transformer uses the latent space of the VQ-GAN with resolution $m_1 \times m_1$ (latent space with higher compression factor).

8.6.2 Super-Resolution Transformer

When training the Super-Resolution Transformer, we assume that we have already completed the Base Transformer's training. The Super-Res Transformer is a MaskGST (Section 8.3.3), like the Base Model, but with a higher resolution latent space ($m_2 \times m_2$). What is different in this model is its input.

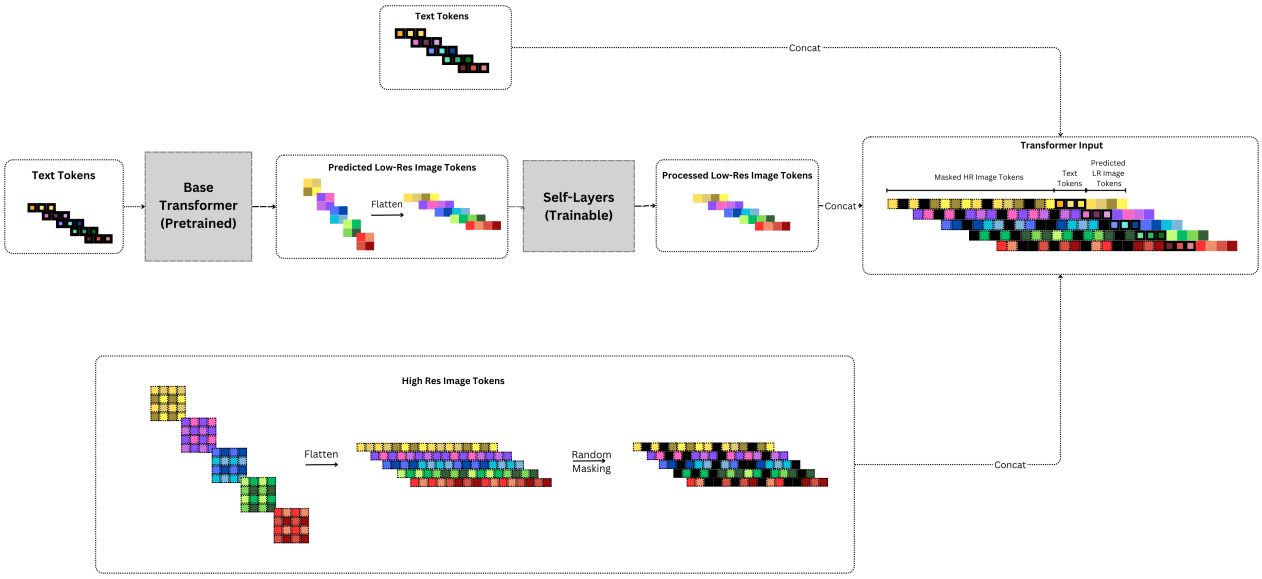


Figure 8.6.1: The Super-Resolution Model's input is formed by concatenating the Masked High-Resolution(HR) image tokens, with the text tokens and the Low-Resolution(LR) image tokens, predicted by the Base Model

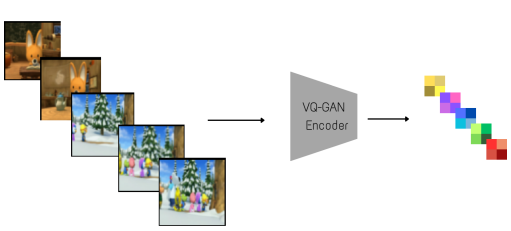
Given the High Resolution image tokens $Z = \{Z_1, Z_2, ..., Z_n\}$ ($Z_i \in \mathbb{R}^{m_2 \times m_2 \times d}$), predicted by the respective VQ-GAN, the text tokens $T = \{T_1, T_2, ..., T_n\}$ ($T_i \in \mathbb{R}^{l \times d}$) and the Low Resolution image Tokens, $Z^{LR} = \{Z_1^{LR}, Z_2^{LR}, ..., Z_n^{LR}\}$ ($Z_i^{LR} \in \mathbb{R}^{m_1 \times m_1 \times d}$), we form the transformer's input as shown in Figure 8.6.1. The image tokens $Z \in \mathbb{R}^{n \times m_2 \times m_2 \times d}$ are flattened into a sequence $Z' \in \mathbb{R}^{n \times (m_2 \cdot m_2) \times d}$. Then they are randomly masked, as in *MaskGIT* to obtain $\bar{Z} \in \mathbb{R}^{n \times (m_2 \cdot m_2) \times d}$. The predicted Low Resolution Tokens are likewise flattened and passed through a series of Self-Layers (following MUSE). Finally, the Masked HR image tokens, the text tokens and the predicted LR image tokens are concatenated to form: $Input_i = (\bar{Z}_i; T_i; Z_i^{LR})$. The Transformer's input can be written as :

$$Input = \{Input_1, ..., Input_n\} \in \mathbb{R}^{n \times (m_2 \cdot m_2 + l + m_1 \cdot m_1) \times d} \tag{8.6.1}$$

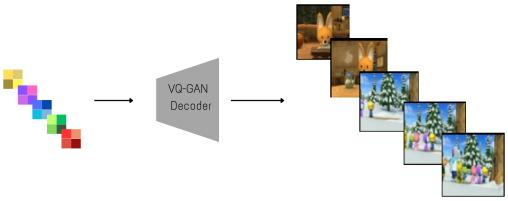
where l is the length of a captions text representation and d is the Transformer's hidden dimension.

8.7 Latent Space Disentanglement

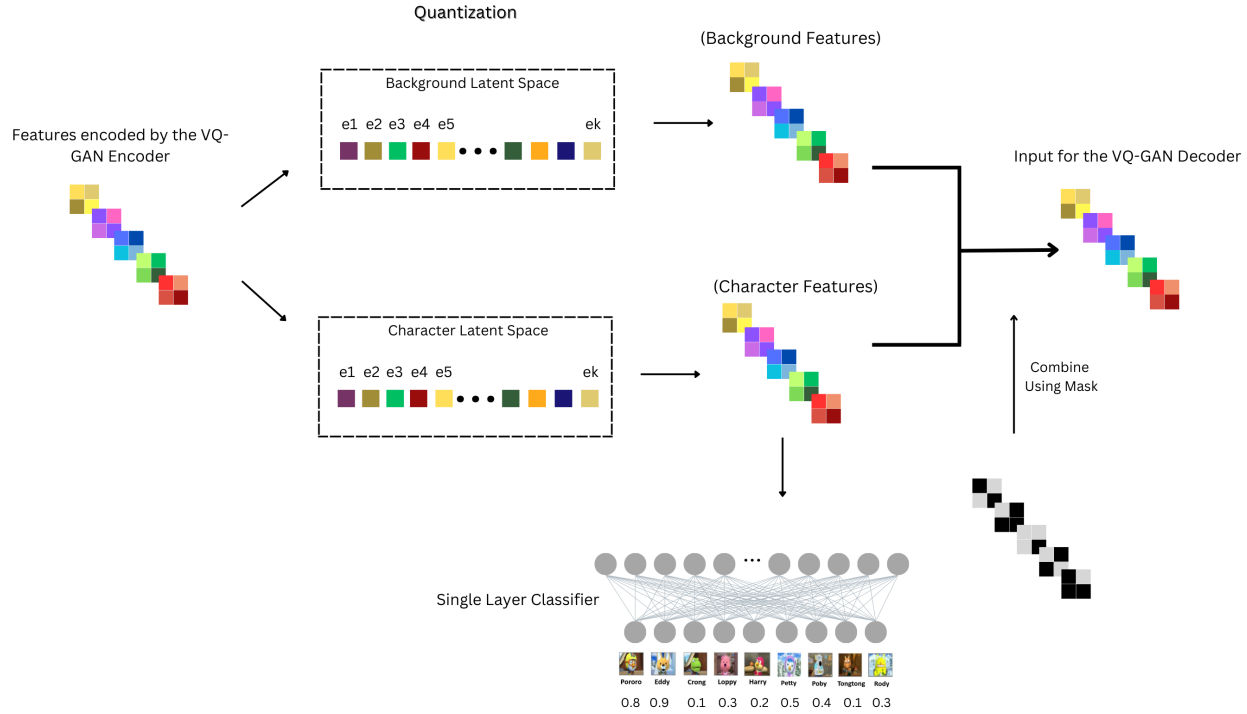
With the improvement of character generation in mind, we propose a technique that attempts to disentangle the latent features that map to characters from the rest. To that end, we modify the VQ-GAN architecture, to add a second library of discrete latent vectors.



(a) The VQ-GAN encoder remains untouched



(b) The VQ-GAN decoder remains the same, as well



(c) We modify the quantization stage, trying to achieve feature disentanglement

Figure 8.7.1: Modified VQ-GAN for feature disentanglement

8.7.1 VQ-GAN Encoder and Decoder

As Figures 8.7.1a and 8.7.1b show, the Encoder and the Decoder of the VQ-GAN remain the same. As we have described previously, they are both CNNs. the Encoder maps an image to a latent space: $x \in \mathbb{R}^{3\times N\times N} \longrightarrow z \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}}$, where D is the dimensionality of the Latent Space and f is a compression factor. Symmetrically, the Decoder maps a point in latent space to a point in pixel space (image): $z \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}} \longrightarrow x \in \mathbb{R}^{3\times N\times N}$

8.7.2 Quantization with two Libraries of Latent Vectors

Instead of using a single library of latent vectors, as in the original VQ-GAN architecture, we use two: $e^{background} \in \mathbb{R}^{K\times D}$ and $e^{char} \in \mathbb{R}^{K\times D}$, where K is number of discrete embeddings in each library and D is dimensionality of the Latent Embeddings. The high-level idea is that $e^{background}$ encodes features that correspond to the background of an image, whereas e^{char} is concerned with character features. An overview of the Quantization process can be seen in Figure 8.7.1c.

An image $x \in \mathbb{R}^{3\times N\times N}$ is first passed through the encoder to obtain $z_0 \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}}$. We apply a D-dimensional filter $f_{background}$ to z_0 and obtain $z_1 = f_{background}(z_0)$. Then we quantize z_1 according to $e^{background}$, by substituting each one of the $(\frac{N}{f} \times \frac{N}{f})$ vectors (each vector has D dimensions) by their closest one in the library. The result of the quantization is $z_{background} \in \mathbb{R}^{D\times \frac{N}{f}\times \frac{N}{f}}$.

Similarly, we use another D-dimensional filter f_{char} to obtain $z_2 = f_{char}(z_0)$. Then we quantize z_2 according to e^{char} and obtain $z_{char} \in \mathbb{R}^{D \times \frac{N}{f} \times \frac{N}{f}}$.

We combine z_{char} and $z_{background}$ using a background mask. Let $M \in \mathbb{R}^{\frac{N}{f} \times \frac{N}{f}}$ be the background mask, with:

$$M_{i,j}^{background} = \begin{cases} 0 & \text{if the (i,j) region in the original image belongs to the foreground} \\ 1 & \text{if the (i,j) region in the original image belongs to the background} \end{cases} \tag{8.7.1}$$

Where regions with Characters are considered to belong to the foreground, whereas regions without characters are considered to belong to the background. We broadcast $M^{background}$ from $M^{background} \in \mathbb{R}^{\frac{N}{f} \times \frac{N}{f}}$ to $\hat{M}^{background} \in \mathbb{R}^{D \times \frac{N}{f} \times \frac{N}{f}}$. The final latent representation of the image if formed as follows:

$$z = z_{background} \odot \hat{M}^{background} + z_{char} \odot \overline{\hat{M}^{background}} \tag{8.7.2}$$

That is, for the regions that belong to the background, $z_{background}$'s embeddings are used, whereas for regions of the foreground (character regions) z_{char}'s embeddings are used. z can be fed into the Decoder to produce the reconstructed image.

As Figure 8.7.1c shows, the character latents z_{char} are also passed through a single-layer Neural Network. This network takes in a flattened version of z_{char}, $\tilde{z}_{char} \in \mathbb{R}^{D \cdot \frac{N}{f} \cdot \frac{N}{f}}$ and outputs class probabilities for all the main Characters in the dataset (9 for Pororo-SV). We train this simple network under a Multi-Label classification loss, using character references in the corresponding captions as ground-truth. The purpose of this network is to provide further feedback to the character latent library e^{char} during training. By using a single layer Neural-Network for classification we hope that the model will be forced to promote disentanglement in the latent embeddings that encode different characters, themselves, because the classifier is supposedly too short to perform the disentanglement in its entirety.

8.7.3 Foreground-Background Segmentation

In order to obtain the background Masks we follow the approach of [5], that we briefly outlined in Section 3.5.2. We use a pretrained convolutional classifier, fine-tuned to detect Characters in our dataset's images. We apply GradCAM [31], using the classifier, to produce a heatmap for each image. GradCAM provides heatmaps that highlight image areas that are highly attended to when the image is passed through the classifier. In our case, these areas are the Character regions in the image. The "relevance" of an image region when taking the Classification decision is given as a real number. Therefore, GradCAM outputs a heatmap: $H \in \mathbb{R}^{N \times N}$, where image regions that are highly attended to, during classification typically have greater values. By choosing an activation threshold h, we can easily obtain a background mask, from H as follows:

$$M_{i,j}^{background} = \begin{cases} 0 & \text{if } H_{i,j} \geq h \\ 1 & \text{if } H_{i,j} < h \end{cases} \tag{8.7.3}$$

where a mask value of 1 (masked out) corresponds to areas that do not contain Characters.

8.7.4 Modifications in the Transformer

The Transformer used for this method is very similar to the ones we have already discussed, with some small modifications.

Training

As we described in Section 8.3.3, after dropping the text part of the transformer's output we get the following result:

$$\overline{Output} = \{\overline{Output_1}, ..., \overline{Output_n}\} \in \mathbb{R}^{n \times (m \cdot m) \times d} \tag{8.7.4}$$

Originally, we use one linear transformation, to turn these outputs into logits, which are enough to predict the image tokens in the latent space of the original VQ-GAN.

However, when using a VQ-GAN with two Latent Libraries, as we described above, we need the transformer to predict the following, in order to generate an image:

- Character logits (and thus Character Tokens)
- Background logits (and thus Background Tokens)
- Background mask.

In order to achieve this, we introduce three linear transformations in the Transformer (instead of one): **To_Char_Logits**, **To_Background_Logits**, **To_Mask**. Where:

$$\begin{aligned} y^{char} &= \textbf{To\_Char\_Logits}(\overline{Output}) \in \mathbb{R}^{n\times(m\cdot m+l)\times K} \\ y^{background} &= \textbf{To\_Background\_Logits}(\overline{Output}) \in \mathbb{R}^{n\times(m\cdot m+l)\times K} \\ y^{mask} &= \textbf{To\_Mask}(\overline{Output}) \in \mathbb{R}^{n\times(m\cdot m+l)\times 1} \end{aligned} \tag{8.7.5}$$

The loss function is a slightly modified version of MaskGIT's loss:

$$\mathcal{L} = \mathcal{L}_{char} + \mathcal{L}_{background} + \mathcal{L}_{mask}, \tag{8.7.6}$$

where:

$$\begin{aligned} \mathcal{L}_{char} &= -\mathbb{E}[\sum_{\forall i\in[1,N], m_i=1, m_i^{background}=0} \textbf{log}p(y_i^{char}|Y_{\bar{M}})] \\ \mathcal{L}_{background} &= -\mathbb{E}[\sum_{\forall i\in[1,N], m_i=1, m_i^{background}=1} \textbf{log}p(y_i^{background}|Y_{\bar{M}})] \\ \mathcal{L}_{mask} &= \mathcal{L}_{CE}(y^{mask}, M^{background}) \end{aligned} \tag{8.7.7}$$

Y stands for the indices of the embeddings in z, where z is given by Equation 8.7.2, whereas $Y_{\bar{M}}$ is the masked version of the indices, according to MaskGIT's masking schedule. Note that the random mask M that is used here is not to be confused with the Background mask $M^{background}$. $\mathcal{L}_{char}$ is the MaskGIT style loss for the character logits, in Character regions of the image. $\mathcal{L}_{background}$ is the MaskGIT style loss for the background logits, in background regions of the image. $\mathcal{L}_{mask}$ is the cross-entropy loss between the ground-truth and predicted Background mask.

Except for the changes described above, the rest of the model is identical to our initial MaskGST.

Inference

During inference we need to generate the Background Mask, the Character Logits and the Background Logits. We start by generating the Background Mask. We do this with a single pass through the transformer. For this pass, all visual tokens in the input are masked. We get an output as described by Equation 8.7.4. Finally, we form $\sigma(\textbf{To\_Mask}(\overline{Output})) \in \mathbb{R}^{n\times m\times m}$, where $\sigma(.)$ stands for the sigmoid function. The Background Mask can be predicted as:

$$\tilde{M}_{i,j,k}^{background} = \begin{cases} 1 & \text{if } \sigma(\textbf{To\_Mask}(\overline{Output}))_{i,j,k} \geq 0.5 \\ 0 & \text{if } \sigma(\textbf{To\_Mask}(\overline{Output}))_{i,j,k} < 0.5 \end{cases} \tag{8.7.8}$$

After predicting the Background Mask, we perform iterative parallel decoding as in MaskGIT. The only difference is that at each step we predict Character Logits and Background Logits. Then the logits are formed as :

$$Logits_{i,j,k} = \begin{cases} Logits_{i,j,k}^{Background} & \text{if } \tilde{M}_{i,j,k}^{background} = 1 \\ Logits_{i,j,k}^{Char} & \text{if } \tilde{M}_{i,j,k}^{background} = 0 \end{cases} \tag{8.7.9}$$

Chapter 9

Experimental Section

9.1 Experimental Setup

9.1.1 Codebase

Our entire codebase is developed in PyTorch. The code for the MaskGIT models is adapted from an open source implementation of MUSE[1]. For image tokenization, we use the original VQ-GAN implementation from *Taming Transformers* [8], that is available on github[2]. All experimental modifications of the VQ-GAN architecture are also based on this implementation. The BPE tokenizer that we use is built using the *Tokenizer* class from Hugging Face[3].

9.1.2 Training Environment

All experiments are performed on the training environment provided by GRNET on ARIS. Specifically, we perform all training and inference on a sinlge NVIDIA V100-16GB GPU.

9.1.3 Story Visualization Datasets

Pororo-SV

Figure 9.1.1: Main characters featured in Pororo-SV

The main dataset used in literature, for Story Visualization is Pororo-SV, proposed in [18]. It is based on a cartoon series names "Pororo the Little Penguin". Pororo-SV was, in fact adapted from a Video-QA dataset [14] based on this series. The original QA dataset comprised of short videos that where associated with text descriptions. In order to adapt the dataset for story visualization, the text descriptions are used as the text input for stories. Additionaly, a frame is sampled from each short video and is associated with the corresponding text description, as its ground-truth image. Five continuous text-image pairs form a story. The Pororo-SV dataset contains 15,336 stories in total. The split used in the original StoryGAN paper contained 13,000/2,336 train/test stories. However, there was overlap between the images used in the training and test sets, that is not considered good practice. Therefore, we adopt the split proposed in [20], which comprises

[1] https://github.com/lucidrains/muse-maskgit-pytorch
[2] https://github.com/CompVis/taming-transformers
[3] https://huggingface.co/learn/nlp-course/chapter6

of 10191/2334/2208 train/validate/test stories. There is no overlap between the train and test sets in this split. The dataset features 9 recurring characters that can be seen in Figure 9.1.1

Flintstones-SV

The Flintstones-SV dataset was proposed in [19]. It was adapted from a Text-to-Video Dataset. A single frame is sampled from each video clip and frames from adjacent clips are gathered into 5-frame stories, similar to what we have in Pororo-SV. Flintstones-SV features 7 major recurring characters and has 20132/2071/2309 training/validation/test samples.

9.1.4 Story Visualization Metrics

FID

We evaluate the quality of the generated images using **FID**. To compute FID we use an pretrained Inception v3 (a CNN) to extract features from images. After extracting the features of the generated images and the ground truth ones, we compares their means and standard deviations. Deep level features extracted from a CNN are expected to correspond to high-level real-world concepts and are, therefore considered to be a better ground for comparison, compared to straight comparison of image pixels.

Character-F1

Character-F1 score is proposed as a metric in [20]. They fine-tune a pretrained Inception v3 model with multi-label classification loss, to identify the presence of main characters in images. This model is then used to predict the presence of characters in generated images. Using the reference of the characters' names in captions as ground-truth we can calculate a models F1-score on character generation. Since we have multiple characters, a micro-average F1-score is calculated. For the sake of consistency we use the same fine-tuned Inception v3, that is publicly available.

Character Accuracy

Similarly to Character-F1, [20] proposes to use the fine-tuned model that was mentioned above, to calculate a models character generation accuracy (Character-Accuracy). The three metrics mentioned above have been widely adopted by papers that study the Task of Story Visualization.

BLEU-2/3

In [20] additional evaluation method is proposed to better capture the narrative element of the task. Specifically, they fine-tune a video captioning model and use it to produce a single text-description for an image-story. They produce story captions using the generated image-stories as well as the ground-truth ones. Finally, BLEU evaluation is performed between the generated and ground-truth descriptions. This method is not as widely adopted as the previous ones, while several different video captioning models have been used in different works. However, we choose to adopt the original video captioning model from [20] and report BLEU-2/3 scores as a supplementary evaluation metric.

9.2 Architectural Experiments

9.2.1 Image Tokenizer

VQ-GAN

During early experimentation we test the following compression factors for VQ-GAN's latent space: $f = 4$, $f = 8$ and $f = 16$, that correspond to latent resolutions of sizes 16×16, 8×8 and 4×4 respectively (all images in the dataset are of size 64×64). The size of the Latent Space is set to 128×256, i.e. 128 discrete latent embeddings, with 256 features each, for all models. All models have $\sim 35M$ parameters.

As our intuition would suggest, we found that lower compression factors lead to better image reconstruction when training the VQ-GAN model. That is, for $f = 4$ we obtain better reconstruction FID compared to $f = 8$ and $f = 8$ achieves better reconstructions than $f = 16$. However, when combining VQ-GAN with our transformer models to predict image tokens from text, we found that this is not the case. When paired with a transformer, the VQ-GAN with $f = 8$ yields the best results, by far, across all metrics.

The above observation might seem counter-intuitive, but we make the following hypothesis: When predicting image tokens from text, our model will find it difficult to reproduce very fine details in the image based on a small text description, that might not even contain these details. Therefore, using a very expressive latent space (low compression factor) might not be beneficial, since the latent tokens will represent finer details in the image and our model will fail to effectively map text into them. On the other hand, a more compressed latent space will be composed of tokens that encode "coarser" characteristics in the images. We hypothesise, that such tokens are more suitable to predict from text descriptions that lack fine details. This could explain why a more compact latent space ($f = 8$) might outperform a more expressive one ($f = 16$) when predicting images from text. Of course, if we choose to use too high a compression factor, there will unavoidably be a decline in the generated image quality at some point, as the features encoded in the latent space will get too "coarse". This could explain why our VQ-GAN with $f = 8$ performs better that the one with $f = 16$.

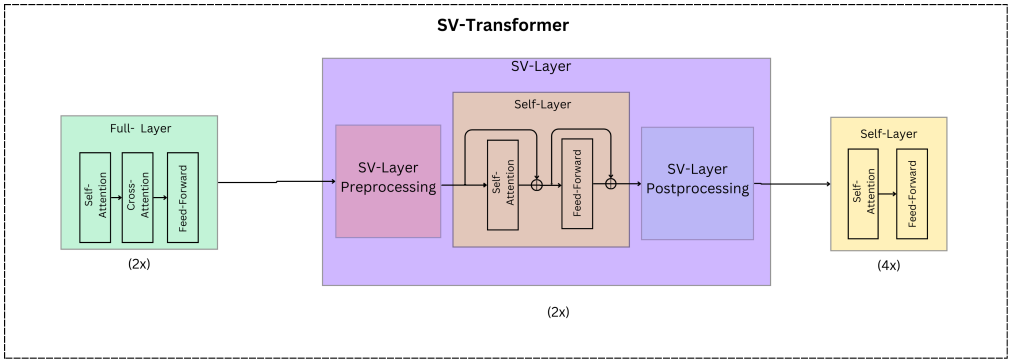
9.2.2 MaskGST

As we mentioned in Section 8.3.3 the vanilla MaskGST is composed of two Full-Layers followed by several Self-Layers. For our experiment, we choose the number of Self-Layers to be equal to 4, which translates into 6 layers in total for the Transformer. The model is identical to example shown in Figure 8.3.5. The Transformer Dimension is set to $d = 1024$. The Vocabulary Size for the text embeddings is set to $n_{vocab} = 2500$. We use Scaled Dot-Product Attention and the number of attention heads is set to $n_{heads} = 8$. The Transformer has $70M$ parameters and the entire *MaskGIT* model (including the VQ-GAN) has 105M parameters. We train the model for 200 Epochs with a learning rate of $lr = 5e - 3$. As we can see in Table 9.6, MaskGST outperforms previous GAN architectures in all metrics. It also, closely approaches previous Transformer Architectures, even surpassing them in terms of Character Accuracy and BLEU scores.

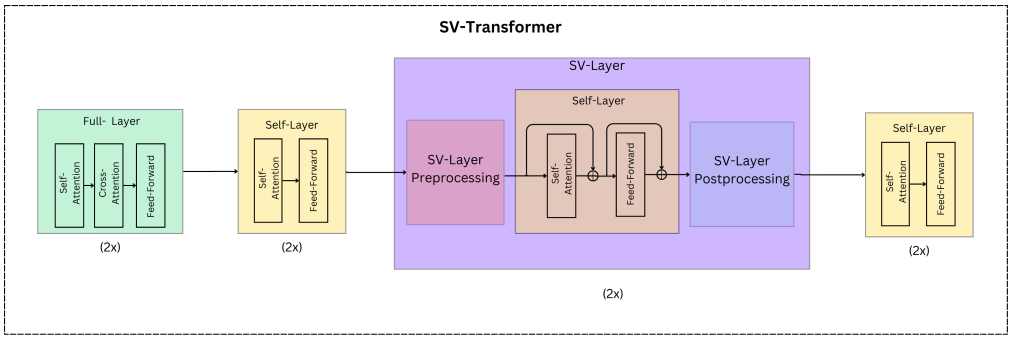
We use this model as our baseline architecture. All next experiments build on this baseline in different ways, in search of ideas that can improve the quality of the generated stories.

9.2.3 MaskGST-SV

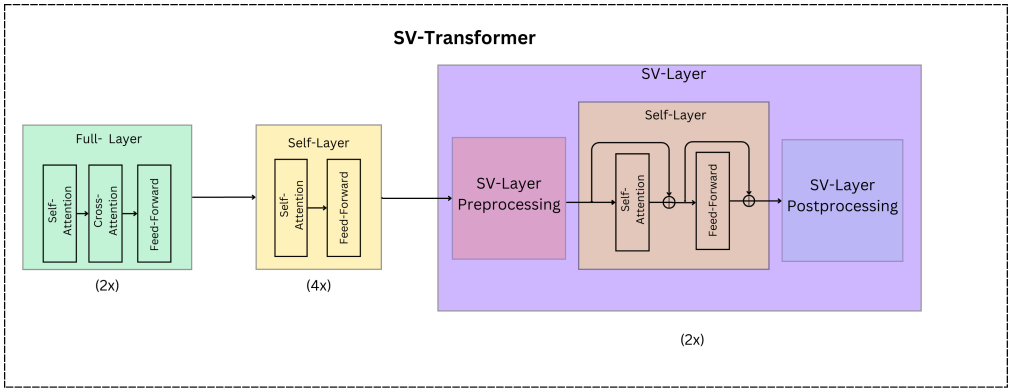
In Section 8.3.3 we described MaskGST-SV as a model that starts with two full-layers, optionally followed by several self-layers, followed by two SV-layers, optionally followed by several extra self-layers. We experiment with three alternative configurations that follow these rules. They can be seen in Figure 9.2.1



(a) MaskGST-SV with the SV-layers positioned early in the generative process



(b) MaskGST-SV with the SV-layers positioned in the middle of the generative process



(c) MaskGST-SV with the SV-layers positioned in the end of the generative process

Figure 9.2.1: Alternative MaskGST-SV architectures

Configuration (a) (Figure 9.2.1a) consists of 2 Full-Layers, followed by 2 SV-Layers, followed by 4 Self-Layers. Configuration (b) (Figure 9.2.1b) consists of 2 Full-Layers, followed by 2 Self-Layers, followed by 2 SV-Layers, followed by 2 Self-Layers. Finally, configuration (c) (Figure 9.2.1c) is comprised of 2 Full-Layers, followed by 4 Self-Layers, followed by 2 SV-Layers. One can easily observe that all three alternatives utilize 2 Full-layers, 4 Self-Layers and 2 SV-Layers, for a total of 8 layers. What changes is the position where the SV-Layers are inserted. Starting from the first configuration and moving towards the third one, the SV-Layers are progressively moved from an early stage in the generative process to later ones.

All hyperparameters are kept the same as in MaskGST ($d = 1024$, $n_{vocab} = 2500$, $n_{heads} = 8$. $lr = 5e-3$, $n_{epochs} = 200l$).

Model	**FID**	**Char-F1**	**Char-Acc**	**BLEU-2/3**
MaskGST	66.12	50.48	26.12	4.68/2.01
MaskGST-SV (a)	63.73	51.83	26.50	4.83/2.17
MaskGST-SV (b)	62.60	52.48	26.65	4.74/1.96
MaskGST-SV (c)	76.34	42.68	18.94	4.27/1.84

Table 9.1: Experimental Results for MaskGST-SV

In Table 9.1 we gather the results for all three configurations. We also include the vanilla MaskGST's results as a reference. We observe that configurations (a) and (b) bring improvement across all metrics compared to the vanilla version. Specifically, configuration (b) achieves the largest improvement upon FID, Char-F1 and Char-Acc, which we consider our principal metrics (in terms of BLEU scores, config (a) is slightly better). Configuration (a) places the SV-Layers immediately after the first 2 Full-Layers, compared to (b) that places them later on the generative process, by inserting 2 Self-Layer between the Full-Layers and the SV-Layers. Full-Layers and Self-Layers treat the visual tokens of each image separately. Therefore, (b) that employs more such layers before the SV-Layers allows the visual tokens to be formed by there local context in a deeper way, before the SV-layers that treat all tokens from the story as a continuous sequence. We hypothesize that letting image tokens be forged by local attention through more layers, before getting into the SV part of the Transformer in configuration (b) is what makes it prevail compared to (a).

Contrary to configurations (a) and (b), configuration (c) yields significantly worst results (higher FID and lower Char-F1, Char-Acc and BLEU scores) compared to the other configurations and even compared to the vanilla MaskGST. Our results point out that SV-Layers should definitely not be placed last in the generative process. That is, they should be followed by several other layers that deal with visual tokens of each image separately (Self-Layers).

In general, our results make it evident that the inclusion of SV-Layers in the generative process for the task of Story Visualization can be beneficial. However, the position of these layers is of great importance and should be subject to careful tuning for a specific model.

9.2.4 T5-XXL as a Text Encoder

To experiment with T5-XXL as a Text Encoder, we use MaskGST, without its text embeddings. Instead of using them, we get an encoding for each caption by passing it through a pretrained T5-XXL model and using its last hidden states as input text embeddings for our Transformer. In Table 9.3 we can see that this approach (MaskGST w/ T5-XXL) does not significantly improve any metric compared to the vanilla MaskGST. In fact, FID is slightly raised, while Char-Acc and BLEU scores are reduced. Only Char-F1 improves by a small margin.

This comes against the intuition that a strong text encoder will provide the model with more expressive text embeddings, that will supposedly lead to an improvement in image token generation. However, we argue that this is to be expected because the language corpus of our datasets significantly differs from the everyday language that is supposed ot be best understood and encoded by a text model. Specifically, the most significant words in our corpus are arguably the names of the Characters (e.g. Pororo, Crong etc), most of which are not "real" words of the English Language. It follows that a text model, like T5-XXL, will probably not be primed to encode them with deep meaning, since they belong to a very niche type of corpus. On the contrary, training text embeddings from scratch, as we do in our original MaskGST, allows to form a specific embedding for each one of the Character names (and any other significant word in our dataset) and forge it in a way that is precisely beneficial for our image generation objective.

9.2.5 Caption Set Augmentation via ChatGPT

The model used in this experiment is identical to the MaskGST that is used in our initial experiment. The only thing that changes is that during training we randomly pick between the original caption and the ChatGPT-generated one for each training sample, at each epoch. During inference we use the original captions of the dataset.

As we can see in Table 9.3, this experiment (MaskGST w/ aug. captions) yields significantly improved compared to the vanilla MaskGST (initial experiment), across all metrics (except for BLEU scores, that are slightly lower). It is of essence to point out the fact that these improvements are solely a result of the augmented training captions, since the model is exactly the same in both cases. We presume that our results confirm our hypothesis, that using multiple captions for each image helps the model focus on the most important concepts (e.g. Character names) and shields against overfitting.

This experiment showcases the ability of LLM's to assist with tasks that they have not been directly optimized for, with remarkable skill. Especially in our case, carrying out text data augmentation following specific instructions, is a very useful automation, since it is a tedious task to ask human annotators to carry out, at a large scale.

9.2.6 Character Guidance

In order to conduct this experiment we use MaskGST and add $2C_n$ Character Embeddings (one positive and one negative per character) as we detailed in Section 8.3.4 ($C_n = 9$ for Pororo-SV). The hyperparameters of the model are the same as in previous experiments ($d = 1024$, $n_{vocab} = 2500$, $n_{heads} = 8$. $lr = 5e-3$, $n_{epochs} = 200$).

When training, we set the text condition drop probability to 20%, that is we completely drop the text embeddings for 20% of the training samples, at each epoch and replace them with a `[NULL]` embedding. For these samples, the Transformer predicts the masked image tokens solely relying on image token self-attention and the Character Embeddings. For the remaining 80% of the training samples, we condition on both text embeddings and Character Embeddings.

At inference time we form text-conditioned logits ℓ_{tc} and character conditioned logits ℓ_{char} as we described in Section 8.3.4. The final logits are given by: $\ell = (1-f)\ell_{tc} + f\ell_{char}$. We choose $f = 0.2$, which is consistent with the text-condition drop probability at training time and is also is proven to be a good choice experimentally.

The results for this experiment can be seen in Table 9.3 (MaskGST-CG$_+$). We observe that this variation of the original architecture has a significant positive impact on all metrics. Our assumption that using explicit character logits, together with the text-conditioned logits will improve Character Generation is clearly confirmed by the significant improvement of both Char-F1 and Char-Acc. It is also important to point out that by doing so we get the added benefit of a surprisingly reduced FID score (66.12 for the Baseline MaskGST vs. 56.78 when adding Character Guidance). BLEU scores are raised as well. It is also worth mentioning that these improvements are achieved without any significant increase in model size. Specifically, the Character Embeddings that we add to the original architecture only have several thousand parameters, which is negligible when added to a model of size in the orders of $\sim 100M$ parameters.

9.2.7 Negative Prompting

For this experiment we use the same model as in the previous one (Character Guidance). Therefore, nothing changes architecture-wise and training-wise. What we modify is the inference scheme. At each iteration of the inference process, we calculate the text-conditional logits and the "logical complement" of the character logits ($\ell_{\overline{char}}$), as we detail in Section 8.3.4. The logits (ℓ) at the current step are then calculated as: $\ell = (1+f)\ell_{tc} - f\ell_{\overline{char}}$.

The results for this experiment can be seen in Table 9.3 (MaskGST-CG$_-$). It is evident that Negative Prompting is beneficial for all metrics. Specifically, in terms of Char-F1 and Char-Acc the improvement is of the same magnitude as in Character Guidance. In terms of FID, we can still see an improvement, albeit more modest than the one achieved by Character Guidance, while BLEU scores deteriorate slightly.

A fuller and more natural way to use Negative Prompting is to combine it with Character Guidance, in order to push the logits towards Characters that need to be present in an image and away from Characters we that need to be absent. To achieve this, we form the logits (ℓ) at each iteration of the inference process as a combination of three independent sets of logits: text-conditional logits (ℓ_{tc}), character logits (ℓ_{char}) and the

"logical complement" of the character logits ($\ell_{\overline{char}}$). We obtain ℓ as follows:

$$\ell = (1 - f)\ell_{tc} + 2f\ell_{char} - f\ell_{\overline{char}} \tag{9.2.1}$$

with $f = 0.2$. Note that the sum of logit coefficients is $(1 - f) + 2f - f = 1$.

The results for this method are in Table 9.3 (MaskGST-CG$_{\pm}$). We observe that the combination of Character Guidance and Negative Prompting is indeed beneficial. The two methods seem to reinforce and complement each other in order to improve the results even more, across the four metrics. This combination of methods yields the best results compared to all previous Architectural experiments (lowest FID and highest Char-F1, Char-Acc and BLEU).

9.2.8 Character-Attentive Token Critic

As we described in Section 8.5, the token Critic is an auxiliary Transformer that utilizes self-attention between visual tokens and cross-attention with trained Character Embeddings to output confidence scores for the visual tokens at each iteration of the inference process. The confidence scores are then used to decide which tokens to keep and which to remask for the next iteration. The transformer that we experiment with can be seen in Figure 9.2.2. It comprises of 4 consecutive Full-Layers. It takes in the output of the generative (MaskGST) Transformer and outputs a number between 0 and 1 for each visual token (its confidence score). We train a Token-Critic with the Transformer Dimension set to $d = 512$. We use Scaled Dot-Product Attention and the number of attention heads is set to $n_{heads} = 8$. The entire model (including the generative Transformer and the VQ-GAN) contains $139M$ parameters. We train the Generative Transformer (MaskGST) and the Token-Critic jointly, for 200 Epochs with a learning rate of $lr = 5e-3$. The Generative Transformer used for this experiment is the vanilla MaskGST.

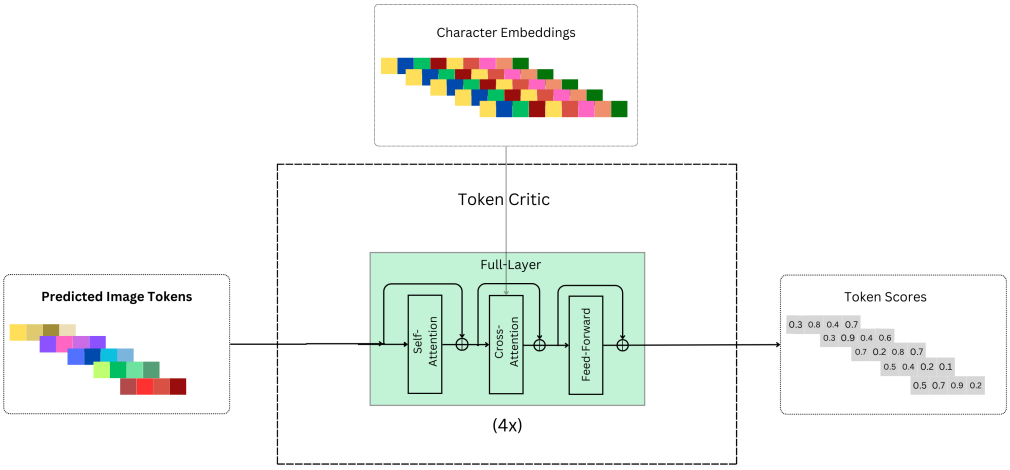


Figure 9.2.2: The Token-Critic takes in the predicted image tokens and uses cross-attention with Character Embeddings to output more informed confidence scores for each visual token in the sequence.

The results for this experiment can be seen in Table 9.3 (MaskGST w/ Char-Attn T.C.). We observe that by using the Token-Critic during inference we face a substantial performance decline across all metrics. One possible explanation for this result is that solely using trained Character Embeddings as a condition for the Token-Critic to predict image token confidence scores is not expressive enough. This could explain why the model fails to distinguish the more suitable tokens from the less suitable ones and therefore leads to a disappointing result.

9.2.9 Latent Super-Resolution Model

We use the vanilla MaskGST that we have already trained for our first experiment as the base Transformer for our Super-Resolution experiment. Our original MaskGST predicts image tokens of an 8×8 resolution, using the latent space of a VQ-GAN with $f = 8$. We train a second Transformer that predicts image tokens

conditioning on text embeddings as well as the low-resolution image tokens predicted by the base model, as we have already described in Section 8.6. This Super-Res Transformer operates in the latent space of a VQ-GAN with $f = 4$ (16×16 resolution).

The Super-Res Transformer that we use is the exact same as our original MaskGST, i.e. 2 Full-Layers, followed by 4 Self-Layers. The Transformer Dimension is $d = 1024$. The Vocabulary Size is $n_{vocab} = 2500$. The number of attention heads is set to $n_{heads} = 8$.The number of training epochs is 200 and the learning rate is set to $lr = 5e - 3$. As Figure 8.6.1 shows, the low-resolution image tokens go through a series of Self-Layers before being used as input to the Super-Res Transformer. We use 4 Self-Layers with $d = 1024$ and $n_{heads} = 8$. The total number of parameters for this model, including the VQ-GAN and excluding the base model is 139M.

In Table 9.3 (MaskGST w/ Latent Super Res.) we can see the results for this experiment. This method yields worst results, in all metrics, even compared to the baseline MaskGST, whose tokens it uses for conditioning. We believe that these findings are further evidence for the remarks we made concerning the VQ-GAN experiments in Section 9.2.1. Specifically, we had made the hypotheses that a Transformer finds it hard to operate in a high-resolution latent space (16×16) because of its fine features that cannot be easily mapped to text-features, using our datasets. We believe this to be - at least in part - the explanation for the failure of this approach as well, since our Transformer performs subpar at a HR latent space, even when conditioning on the LR image tokens of our relatively better (vanilla) MaskGST.

9.2.10 Latent Space Disentanglement

For this experiment we train a modified VQ-GAN with two latent spaces, $e^{char}, e^{background} \in \mathbb{R}^{128\times256}$ as we described in Section 8.7. Using these spaces, the VQ-GAN encodes an image according to the Character Space and the Background Space and combines these encoding according to a Background Mask to form the final latent representation. This representation can be decoded through the Decoder to reconstruct the initial image.

We obtain the Background Mask by applying GradCAM on a classifier that is trained to identify all Characters in the dataset. To that end, we fine-tune a pretrained resnet-50 with a multi-label classification loss on our dataset's training images.

Qualitative Test of Disentanglement

We formulate a qualitative test to get an idea of whether the VQ-GAN achieves the disentanglement it was designed for to some extend.

Let $\mathcal{E}(.), \mathcal{D}(.), \mathcal{Q}^{char}(.), \mathcal{Q}^{background}(.)$ denote the Encoder, Decoder, Character Latent Quantization and Background Latent Quantization functions respectively. As we described in Section 8.7, an image x is encoded into an initial latent representation $z_0 = \mathcal{E}(x)$. z_0 is then mapped into two separate latent spaces (using filters f_1 and f_2) and quantized according to the respective quantization functions: $z_{background} = \mathcal{Q}_{background}(f_1(z_0))$ and $z_{char} = \mathcal{Q}_{char}(f_1(z_0))$. The final latent representation is formed as:

$$z = z_{background} \odot \hat{M}^{background} + z_{char} \odot \overline{\hat{M}^{background}} \tag{9.2.2}$$

where $\hat{M}^{background}$ is the Background Mask. The image can be reconstructed using the decoder: $\tilde{x} = \mathcal{D}(z)$

As we already detailed, the idea behind our method is to push the model towards disentangling high level features of the Characters (e.g. color, shape) from those of the Background of the scenes (e.g. trees, snow). We want character features to be encoded in z_{char} and background features in $z_{background}$. We check whether that is the case, by reconstructing images, only using one of the latent representations. That is, we produce $\tilde{x}_{char} = \mathcal{D}(z_{char})$ and $\tilde{x}_{background} = \mathcal{D}(z_{background})$. Presumably, $\tilde{x}_{char}$ uses the "character features" of the original image, whereas $\tilde{x}_{background}$ uses the "background features" of the original image. Thus we would expect the former to reconstruct Character regions in the image much better that Background regions and vice versa for the later one.

Figure 9.2.3 shows some examples of our test:

- The first example, in Figure 9.2.3a is relatively successful in terms of what we would expect. x_{char} recreates the characters much better than $x_{background}$, especially *Eddy* (the fox that appears in most of the images). On the contrary, $x_{background}$ does not capture *Eddy* very well, but it reconstruct the background much more loyally (e.g. trees, color of the sunset).

- In Example 2 (Figure 9.2.3b), we observe that x_{char} captures the Characters fairly well. It also gets the main features of the background right (e.g. color of the snow, trees in the third and fourth image). On the other hand $x_{background}$ mostly fails to recreate the characters. Especially in the third and fourth panel, it almost removes them. Additionally, if we look closely, it reconstructs the background more faithfully than x_{char}. Specifically, the shape of the clouds in the sky, in the first two panels is more accurate and the mountains behind the trees in images 3 and 4 are present, whereas in x_{char}, they have disappeared.

- Example 3 (Figure 9.2.3c) is less successful. We observe that *Poby* (the white bear) appears in x_{char} as well as $x_{background}$. The background is similar in both reconstructions, although the color of the wall in the fourth and fifth image is captured more accurately in $x_{background}$. Finally, we can see that loopy (the pink beaver in the first image) is reconstructed in x_char, while $x_{background}$ fails to get the color right.

Story Visualization Results

We pair the double-latent-space VQ-GAN with a transformer as described in Section 8.7. The hyperparameters of the Transformer remain the same as in the initial MaskGST. The only difference is that we use three linear projections, instead of one, in order to map the transformer's output into Character Logits, Background Logits and Background Mask.

The results for this experiment (Table 9.3 - MaskGST w/ latent space disentanglement) are underwhelming across all metrics. FID is raised and Char-F1 and Char-Acc are lowered compared to the initial MaskGST. We assume that the Transformer's failure could be explained by the complexity of its modeling task. Specifically, in this experiment, we expect it to model both $p_\theta(z_{char}|T)$ and $p_\theta(z_{background}|T)$, where θ represents the Transformer parameters and T represents the text inputs. Since z_{char} and $z_{background}$ are, in this case, modelled in different latent spaces, maybe it becomes infeasible for the Transformer to learn the two distributions jointly.

9.2.11 Combining Methods

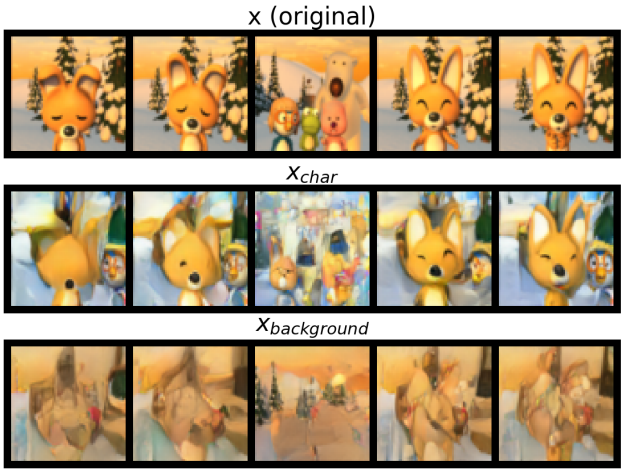
Through the previous experiments we identified several ideas that brought promising results. In this section, we test whether methods that worked well independently can be combined to reinforce each other and further improve our results. Specifically, there are three methods that achieved improvements compared to our baseline MaskGST:

- MaskGST-SV (config b)

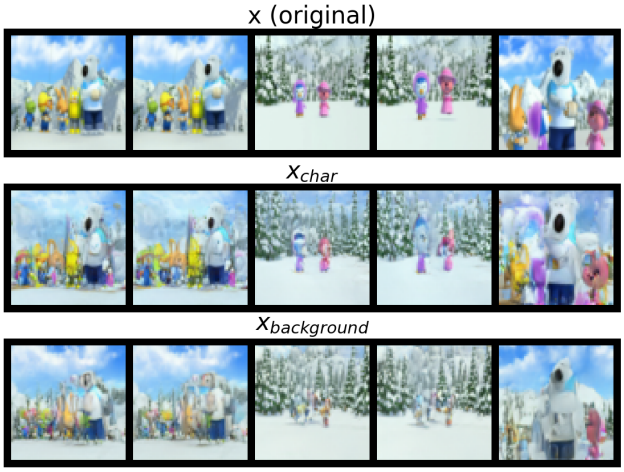
- MaskGST with caption augmentation

- MaskGST-CG$_\pm$ (Character Guidance and Negative Prompting)

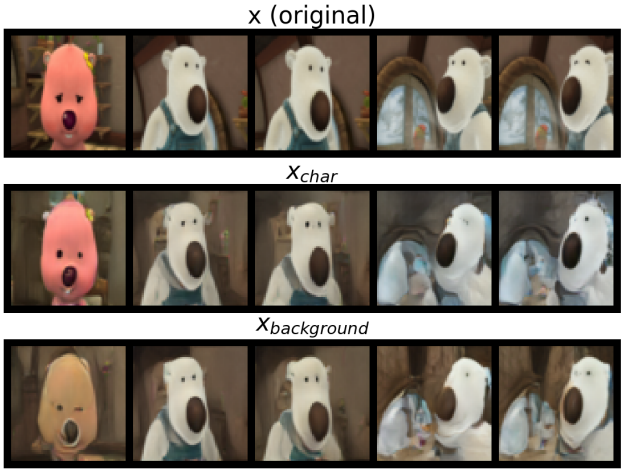
We combine the above methods, in pairs and then all three together. In Table 9.2 we gather the results for these experiments. We also include the results for the Baseline MaskGST and the independent experiments for ease of comparison.



(a) Example 1



(b) Example 2



(c) Example 3

Figure 9.2.3: Examples of our disentanglement test

Model	N-params	**FID**	**Char-F1**	**Char-Acc**	**BLEU-2/3**
(baseline) MaskGST	105M	66.12	50.48	26.12	4.68/2.01
MaskGST-SV (config b)	127M	62.60	52.48	26.65	4.74/1.96
MaskGST w/ T5-XXL	105M (+t5-xxl=770M)	66.63	51.18	25.86	4.55/1.85
MaskGST w/ aug. captions	105M	59.91	54.64	28.67	4.45/1.81
MaskGST-CG$_{+}$	105M	56.78	55.35	29.80	4.91/2.06
MaskGST-CG$_{-}$	105M	62.33	55.27	30.06	4.53/1.89
MaskGST-CG$_{\pm}$	105M	**54.95**	**59.55**	**33.64**	**4.96/2.10**
MaskGST w/ Latent Super Res.	136M	87.55	48.04	23.17	4.21/1.64
MaskGST w/ Char-Attn T.C.	139M	77.98	42.20	19.12	3.51/1.39
MaskGST w/ latent space disentanglement	105M	68.42	43.28	20.67	3.90/1.59

Table 9.3: Experimental Results

Comment	Model	N-params	**FID**	**Char-F1**	**Char-Acc**	**BLEU-2/3**
Initial baseline	Baseline MaskGST	105M	66.12	50.48	26.12	4.68/2.01
Individual Experiments	MaskGST-SV (config b)	127M	62.60	52.48	26.65	4.74/1.96
	MaskGST w/ aug. captions	105M	59.91	54.64	28.67	4.45/1.81
	MaskGST-CG-$_{\pm}$	105M	54.95	59.55	33.64	4.96/2.10
Pairing Individual Experiments	MaskGST-SV + aug. captions	127M	58.54	54.31	28.17	4.83/2.04
	MaskGST-SV-CG$_{\pm}$	127M	60.40	57.75	32.26	4.82/1.97
	MaskGST-CG-$_{\pm}$ w/ aug. captions	105M	51.65	60.46	33.62	4.82/2.00
Combining All Individual Experiments	MaskGST-SV-CG$_{\pm}$ w/ aug. captions	127M	56.89	60.05	33.54	4.83/1.98

Table 9.2: Combining Different Methods

We observe that the best combination is *MaskGST-CG-$_{\pm}$ w/ aug. captions*. Specifically, it improves FID by more than 3 units and Char-F1 by almost one unit, compared to *MaskGST-CG-$_{\pm}$*. Char-Acc remains the same.

It turns out that combining the *MaskGST-SV* with *character guidance & negative prompting* is not beneficial. We assume that this is the case because the later method explicitly encodes the presence or absence (Positive and Negative Embeddings) of Characters in the input that corresponds to each image. However, the MaskGST-SV allows the visual tokens of an image to attend to the inputs for all images in the story (not just their own). Since a Character might be present in an input caption and absent in another caption of the same story, the visual tokens of an image inside MaskGST-SV will, thus end up attending to both Positive and Negative Embeddings for this Character, which is probably confusing for the model.

9.3 Experiments on Hyper-Parameters

9.3.1 Transformer Hyper-Parameters

After having arrived at our best Architecture: *MaskGST-CG-$_{\pm}$ w/ aug. captions*, we perform two non-exhaustive experiments on its Hyper-Parameters. Namely, we explore the effect of the **length** (number of layers) and **width** (hidden dimension) of the Transformer, on our results.

Table 9.4 shows the results for the experiments in terms of length. We keep the other hyper-parameters unchanged (Transformer dimension is $d = 1024$) and scale its length from 4 layers up to 16 layers. We observe that increasing the length of the model results in improvement in all metrics.

In Table 9.5, we gather the results regarding the Transformer's dimension (width). For these experiments we keep all other hyper-parameters constant and vary the dimension from $d = 768$ to $d = 2048$. The number

Transformer Length	N-params	**FID**	**Char-F1**	**Char-Acc**	**BLEU-2/3**
{2 Full-Layers} + {2 Self-Layers}	85M	56.40	57.88	31.01	4.80/2.00
{2 Full-Layers} + {4 Self-Layers}	105M	51.65	60.46	33.62	4.96/2.10
{2 Full-Layers} + {14 Self-Layers}	210M	47.76	63.37	34.92	5.03/2.13

Table 9.4: Experiments on the Transformer Length of *MaskGST-CG*$_{\pm}$

Transformer Dimension	N-params	**FID**	**Char-F1**	**Char-Acc**	**BLEU-2/3**
768	79M	54.90	58.21	31.65	4.90/2.05
1024	105M	51.65	60.46	33.62	4.96/2.10
2048	276M	42.86	65.10	37.50	5.13/2.26

Table 9.5: Experiments on the Transformer Dimension of *MaskGST-CG*$_{\pm}$

of layers in these models is 6 (2 Full-Layers followed by 2 Self-Layers). Increasing the width of the model is beneficial across all metrics. In fact, the Transformer with $d = 2048$ is our top-performing model.

9.3.2 Study on Character Guidance

Using our top model, *MaskGST-CG*$_{\pm}$ *w/ aug.captions* (d=2048), we perform a study, on the Character Guidance factor (f) (Equation 8.3.8). Our results are summarized in Figure 9.3.1. The curves that represent the Character Metrics have the exact same shape. They are increasing, up until $f = 0.6$ and decrease slightly for $f = 0.8$. FID, on other hand decreases (improves) until $f = 0.4$ and then it increases for the next two experiments. At an intuitive level, what we see is that by increasing f we improve the generation of Characters. This is to be expected, since, by increasing f, we pay more attention to the Character logits vs the text-conditional logits, when forming the images. By improving the generation of Characters, we also improve the overall quality of the images, to a certain extend, since Characters are a significant part of them. This is why the increase of f up until the value of 0.4 improves FID (image quality). However increasing f more than that has an adverse effect on overall image quality, since we weigh Character logits disproportionately highly and neglect text-conditional logits, which leads the model to fail in creating other aspects of images, except for Characters.

For $f = 0.4$ we get the optimal combination of metrics (68.32, 41.40 and 42.49 for Char-F1, Char-Acc and FID, respectively). However, in practice, we find that even $f = 0.4$ makes the model focus too much on Character generation and has an adverse effect on the overall quality of the image-story, by impairing the coherence between images. We find $f = 0.2$ to achieve the best trade-off between Character Generation and Story Quality, at a qualitative level.

Model Family	Model	N-params	**FID**	**Char-F1**	**Char-Acc**	**BLEU-2/3**
GAN	StoryGAN	-	158.06	18.59	9.34	3.24/1.22
	CP-CSV	-	149.29	21.78	10.03	3.25/1.22
	DUCO	101M	96.51	38.01	13.97	3.68/1.34
	VLC	100M	84.96	43.02	17.36	3.80/1.44
Auto-Regressive Transformers	VP-CSV	-	65.51	56.84	25.87	4.45/1.80
	CMOTA	97M	52.13	53.25	24.72	4.58/1.90
Diffusion Models	AR-LDM	1.5B	16.59	-	-	-
	ACM-VSG	850M	**15.48**	-	-	-
	Causal-Story	-	16.28	-	-	-
MaskGIT Transformers	(baseline) MaskGST	105M	66.12	50.48	26.12	4.68/2.01
	MaskGST-CG$_{\pm}$ (l=6,d=1024)	105M	51.65	60.46	33.62	4.96/2.10
	MaskGST-CG$_{\pm}$ (l=6,d=2048)	276M	42.86	**65.10**	**37.50**	**5.13/2.26**

Table 9.6: Comparison with previous architectures

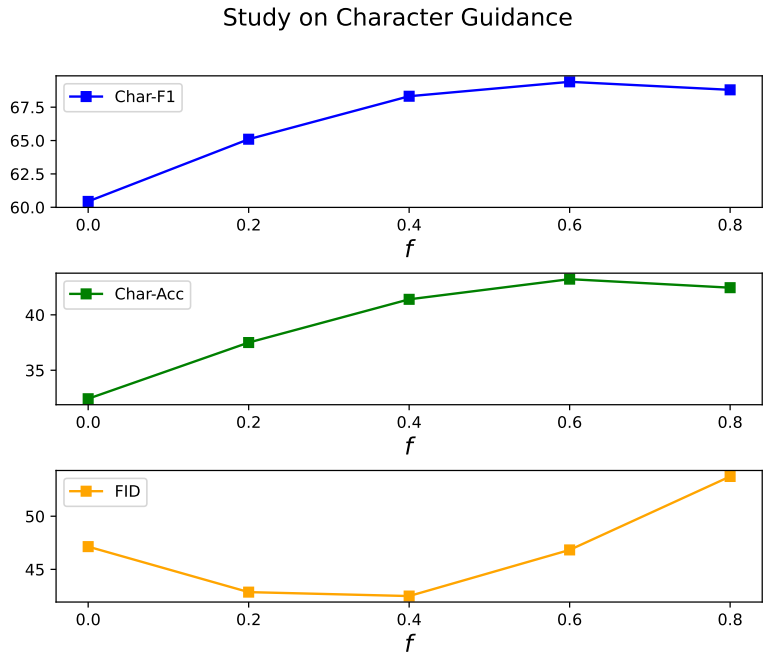


Figure 9.3.1: Study on the Character Guidance factor f.

9.4 Comparison With Previous Baselines

We compare our models to several previous arts. In terms of GAN models, we compare to StoryGAN [18], CP-CSV[36], DUCO-StoryGAN[20] and VLC-StoryGAN[19]. We also compare with two Transformer Architectures; VP-CSV [5] and CMOTA[1]. Regarding Diffusion models, there are three previous arts: AR-LDM[22], ACM-VSG[10] and Causal-Story[35]. Direct comparison with these models is not fair, since they are based on LDM[28], that is pre-trained on a massive dataset with huge compute. Additionally, they use excessively expensive hardware for development. For reference, AR-LDM uses [40×] the vRAM we use. Having said that, we still include these three models in our main comparison, for the sake of completeness. We refer to them separately in Subsection 9.4.3.

Table 9.6 gathers the results of all previous arts, together with our baseline MaskGST and our top-performing

model MaskGST-CG$_{\pm}$, with $d = 1024$ and $d = 2048$.

As we have already mentioned, StoryLDM and StoryGPT-V are applied to a slightly altered version of the task, where repeated mentions of Character names in a story are substituted by pronouns. For this reason, a direct comparison with these models is not applicable.

9.4.1 MaskGST-CG$_{\pm}$ w/ aug. captions ($d = 1024$)

Our model with $d = 1024$ performs better than all previous GAN and Transformer Architectures, in all metrics (lowest FID and highest Char-F1, Char-Acc, BLEU-2/3). Especially Character metrics are raised significantly. Specifically, in Char-F1 there is a 3.6 point improvement, while in Char-Acc we achieve an improvement of 7.7 points, compared to the previous best (VP-CSV). We largely attribute our models superiority in terms of character generation, to our Character Guidance mechanism.

9.4.2 MaskGST-CG$_{\pm}$ w/ aug. captions ($d = 2048$)

Doubling the hidden dimension to $d = 2048$ improves our results across all metrics. Most notably, there is a major improvement in terms of FID, which decreases by 8.8 points compared to the $d = 1024$ version of the model. We assume, that doubling the hidden dimension leaves more room to the model to learn more complex and fine mappings between words and visual features, that result in higher quality images with more details. The model's ability to learn more complex representations and produce more detailed images can account for the improvements in the other metrics as well. Specifically, high quality images will depict improved versions of the characters and produce better BLEU scores, through video captioning.

9.4.3 Comparison with Diffusion models

As it is evident from Table 9.6, all diffusion models only use FID as an evaluation metric. This is insufficient for SV, since it solely takes into account image quality. In terms of FID, all three diffusers are unparalleled, compared to all our models, as well as other previous arts. However this can be attributed-at least to some extend-to their extensive pre-training and large parameter counts.

9.4.4 Story Continuation

[22, 10, 35] also report results for the task of Story Continuation[21]. Story Continuation (SC) is similar to SV. However, in SC the first frame is considered to be given as input and the rest of the frames need to be generated. In Table 9.7 we report SC results for our model ($d = 2048$), as well as AR-LDM[22], ACM-VSG[10] and Causal-Story[35]. Additionally, we include results for StoryDALLE (fine-tune)[21] and Mega-StoryDALLE[21] that are both based on pre-trained auto-regressive transformers. These where the first models applied to the task of SC. We observe that although our model has the lowest parameter count, by far, it outperforms all the other large, extensively pre-trained models in terms of Char-F1 and Char-Acc. This speaks of the remarkable merit of our Character Guidance method. In terms of FID, large pre-trained models remain unparalleled, as in SV. Since our model is trained for SV, in order to evaluate it for SC, we just discard the first generated frame and evaluate it using the remaining four frames.

Model	#param	**FID**(↓)	**Char-F1**	**Char-Acc**
StoryDALLE(fine-tune)[21]	1.3B	25.90	36.97	17.26
Mega-StoryDALLE[21]	2.8B	23.48	39.91	18.01
AR-LDM[22]	1.5B	17.40	-	-
ACM-VSG[10]	850M	**15.36**	45.71	22.62
Causal-Story[35]	-	16.98	-	-
Ours (d=2048)	276M	43.31	**65.32**	**37.38**

Table 9.7: Story Continuation results on the test set of Pororo-SV, for large pre-trained models, as well as MaskGST-CG$_{\pm}$ w/ aug. captions (d=2048) . We report scores for FID (lower is better), as well as Char-F1, Char-Acc (higher is better).

Frame 1: Crong eats meat. Loopy gave a dish of vegetables.
Frame 2: Crong doesn't like vegetables. Crong pushed the dish of vegetables.
Frame 3: Both Pororo and Crong look at the meat in the dish.
Frame 4: Crong picked up the last meat.
Frame 5: Pororo told that Crong is so greedy.

Frame 1: Loopy is back with Poby.
Frame 2: Loopy and Poby notice that chair is not broken.
Frame 3: Poby and Loopy thinks the chair is very strange.
Frame 4: Loopy is looking at chair with question.
Frame 5: Loopy thanks Poby for coming.

Frame 1: Petty asks if Crong really okay is.
Frame 2: Loopy says Crong looks not that good.
Frame 3: Crong trying to pretend to be okay waves Crong head.
Frame 4: Crong pretends to do some freehand exercise.
Frame 5: Pororo laughs at Crong calling Crong as a regular pooping machine.

Frame 1: Loopy talks and spreads Petty arms. Petty looks at Loopy smiles and nods.
Frame 2: Poby gathers red car hands and talks. Loopy Petty and Harry are looking at Poby.
Frame 3: Loopy Poby and Petty walk stop and turn back.
Frame 4: Harry looks angry and talks.
Frame 5: Harry looks at Poby and turns red car head.

Figure 9.5.1: Qualitative Comparison between our model (MaskGST-CG$_{\pm}$ w/ aug. captions) and CMOTA[1] across 4 story examples.

9.5 Qualitative Results

In Figure 9.5.1 we provide four examples of image-stories from the Pororo-SV test set. For each story, we provide its captions, the ground-truth images (Original), the image sequence generated by CMOTA [1] and the one generated by our model (MaskGST-CG$_{\pm}$ w/ aug. captions ($d = 2048$)). To that end, we used the pre-trained CMOTA model that has been released here[4]. We carry out a qualitative comparison based on these examples.

9.5.1 Image Quality

In terms of image quality, it is evident from the examples that our model is superior to CMOTA. For example, in the top-left panel, the first and final images, constructed by CMOTA are blurry and incomprehensible. On the contrary, in our case both images contain discernible objects (characters). In the three middle images, CMOTA's images contain some recognizable characters. However, even in that case, our images are of much higher quality, with the characters having significantly more detailed appearances (e.g. Crong's (the green dinosaur's) eyes and Pororo's (the penguin's) beak).

[4] https://github.com/yonseivnl/cmota

Criterion	Ours (%)	CMOTA (%)	Tie(%)
Visual Quality	**78%**	3%	19%
Temporal Consistency	**66%**	8%	26%
Semantic Relevance	**64%**	9%	27%

Table 9.8: Results of our human survey. We compare the results of our model *MaskGST-CG$_{\pm}$ w/ aug. caption (d=2048)* (Ours) against CMOTA's, across three criteria. Our(%) and CMOTA(%) indicate the percentage of cases where each model was chosen by both annotators, whereas Tie(%) accounts for the remaining cases.

9.5.2 Temporal Consistency

In terms of temporal consistency, we observe that CMOTA's images struggle to hold a consistent background in all 4 examples. Specifically, images from outside are repeatedly interleaved with indoor images. For example, in the top-right panel, the first and final images seem to show a snowy background, whereas the other three look like they are from an indoor space. On the contrary, our model manages to hold a relatively consistent background in most cases. Especially in the top-right panel, the appearance of the room is held exceptionally consistent. We should note, that in the bottom-right panel, our model struggles to be consistent, especially in the fourth image that has an irrelevant background, compared to the adjacent ones.

9.5.3 Semantic Relevance

The term *Semantic Relevance* refers to whether the generated image is relevant to the corresponding caption. With regards to this, CMOTA seems to struggle especially in cases where multiple characters are referenced. For example in the bottom-right panel, the first caption metions Petty and Loopy, but only Petty is generated. In the second caption, where multiple characters are mentioned, CMOTA's image is incomprehensible. In the next three images, CMOTA manages to generate most mentioned characters, albeit with low quality. In contrast, our model manages to generate all relevant characters in most cases. In the bottom-right panel, this is true for all captions. Most notably, in the second image, it manages to generate multiple characters, with remarkable quality, as well as the red car that is mentioned in the caption.

9.6 Human Evaluation

In order to further investigate our qualitative results, we conducted a human survey across three criteria which were also adopted by previous works [20, 19, 1], comparing our model with CMOTA [1]. The three adopted criteria are the following:

- **Visual Quality** refers to whether the images are visually appealing, rather than blurry and difficult to understand.
- **Temporal Consistency** measures whether the images are consistent with each other, having a common topic and naturally forming a story, rather than looking like 5 independent images.
- **Semantic Relevance** refers to whether the images accurately reflect the captions and the characters mentioned in them.

The evaluation is done over 100 stories from the test set of Pororo-SV. Each story is evaluated by 2 distinct annotators. The results of the study (Table 9.8) indicate that our model is superior across all 3 criteria, thus supporting our quantitative results.

9.7 Resource Usage Analysis

9.7.1 Recources at Training

We approximately spend 36 and 107 hours to train our Transformers with $d = 1024$ and $d = 2048$, respectively, on a single NVIDIA V100 (16GB). This is equivalent to (36 hours) $\cdot$ (16GB) $=$ 576 (GB $\cdot$ hours) and (107

hours)· (16GB) = 1712 (GB · hours) of GPU usage, respectively. For reference, VP-CSV [5] reportedly uses 4 NVIDIA A100 (40GB) for 12 hours. This is equivalent to (12 hours) · (4·40 GB) = 1920 (GB · hours) of GPU usage, without taking into account that A100 is a more modern GPU than V100. CMOTA [1] does not report training resource usage.

9.7.2 Recources at Inference

Since we performed inference for our models, as well as CMOTA, on the same GPU, we can carry out a fair comparison. Our models with $d = 1024$ and $d = 2048$ need 34 minutes and 94 minutes, respectively to perform inference for the 2208 stories of the test set. This is equivalent to 0.92 *sec/story* and 2.55 *sec/story*. For the same task, CMOTA spent 228 minutes, which translates to 6.19 *sec/story*. It is evident that our method is significantly more time-efficient compared to CMOTA. It is interesting that even our larger model ($d = 2048$), which has more than double the parameter count of CMOTA, is more that [2×] faster, during inference compared to it. This can be largely attributed to the inference scheme of MaskGIT-style transformers, that produce multiple visual tokens per step, compared to auto-regressive transformers, like CMOTA, that infer visual tokens one at a time.

9.8 More Qualitative Examples

Images 9.8.1 and 9.8.2 show more Story Generation examples using our *MaskGST-CG$_{\pm}$ /w aug. captions*. For each example we provide the images generated by our model, the input captions and the ground-truth images (original) that correspond to these captions.

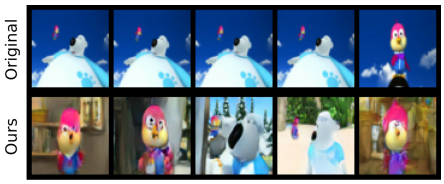
Frame 1: Pororo notices the batter on the table.
Frame 2: Pororo picks up cookie cutter.
Frame 3: Pororo bakes cookies for loopy.
Frame 4: Pororo is finished with baking cookies.
Frame 5: Loopy has come back home.

Frame 1: the mean magician Eddy had to keep singing. the story ends.
Frame 2: Poby is holding a book. Poby says the story is done.
Frame 3: Pororo is telling the lesson with the example of Pororo and Harry.
Frame 4: Roby suggests to help friends.
Frame 5: Poby shook his hand saying good bye.

Frame 1: Pororo and Crong are very curious about what the Robot is. they are curiously watching it.
Frame 2: The robot asks Pororo and Crong what he can do for them.
Frame 3: Pororo and Crong are very surprised to hear the robot talking.
Frame 4: The robot asks again what can he do for Pororo and Crong.
Frame 5: Pororo decides to examine the robot with Crong.

Frame 1: Petty and Loopy are talking in Loopy house.
Frame 2: Loopy points Loopy broken chair.
Frame 3: Poby makes an excuse for Pororo. Loopy is angry.
Frame 4: Loopy explains why Petty is mad at Pororo.
Frame 5: Petty and Poby understands why Loopy is so mad at Pororo.

Frame 1: Pororo makes the snowman's legs. then Pororo makes the arm of the snowman.
Frame 2: Pororo makes arm of the snowman. then Pororo puts goggles to snowman's face.
Frame 3: Pororo puts mouth to the snowman's face. then Pororo puts buttons to the snowman's face to make the snowman's eyes.
Frame 4: Pororo finishes making his own snowman.
Frame 5: Poby is walking on the snow. Poby is waving.



Frame 1: Poby is continually in the air. Harry is now with Poby. Harry says that Harry will make Poby put down.
Frame 2: Harry and Poby are talking each other in the air. Harry tries to make Poby put down.
Frame 3: Harry and Poby are talking each other in the air. Poby has long been on the sky so Poby seems to be tired.
Frame 4: Harry and Poby are talking each other in the air. Harry tries to make Poby happy and make Poby put down. suddenly Harry jumps high and stops on top of Poby's head.
Frame 5: Harry is now on top of Poby's head. harry says that Harry will sing a song for Poby therefore Harry asks to Poby to hang in there.

Figure 9.8.1: More Story Generation Examples using our model *MaskGST-CG$_{\pm}$ /w aug. captions.*

Frame 1: Eddy Loopy and Petty don't want to listen Harry's song. they just want to eat cake.
Frame 2: Harry stands up on the balustrade. Harry tries to sing a song to her friends in spite of the friends' dissuading.
Frame 3: Harry starts to sing a song on the balustrade.
Frame 4: Pororo and his friends looks disgusted. they really don't want to listen harry's song.
Frame 5: Loopy petty and Crong also don't want to listen Harry's song.

Frame 1: Eddy has something with his right hands. that looks like a small fan. its color is light blue. Eddy looks happy.
Frame 2: the front side of the cannon is uncovered with snow. Eddy is walking by following the path which was covered with snow.
Frame 3: Eddy is walking through the path which was covered with snow. Eddy maybe thinks that Eddy will show Pororo what Eddy can do.
Frame 4: while walking Eddy sees that Pororo is coming to him. Eddy stops after watching Pororo.
Frame 5: Eddy is murmuring about something which is related to Pororo. Pororo is coming to Eddy with someone.

Frame 1: Petty would like to give warm tea to Eddy.
Frame 2: Eddy gives a cup of tea to Eddy.
Frame 3: Petty smiles. Eddy said thanks to Petty.
Frame 4: the tea was too salty for Eddy.
Frame 5: Eddy was surprised to hear that there is salt in the tea.

Frame 1: Pororo's friends are shaking their heads because they don't know what Pororo asked.
Frame 2: Harry flies to Pororo and say something.
Frame 3: Pororo ponders something for a moment and Eddy and Petty also walk to Pororo.
Frame 4: Petty and Eddy stand in a row. Eddy says something to Pororo.
Frame 5: Petty and Eddy stand in a row. Petty says something to Pororo and eddy glances at Petty and smiles.

Frame 1: now Pororo is preparing to hit Eddy. Pororo throws snowballs to Eddy. Eddy tries to avoid snowballs.
Frame 2: Eddy looks happy with his eyes wavy shaped. however Eddy is struck snowballs from Pororo.
Frame 3: Pororo seems happy with his successful hitting to Eddy. Pororo is playing snowballs with his right hands. after struck Eddy looks angry.
Frame 4: Eddy is walking toward Pororo. Pororo is looking Eddy with his curious feeling. Pororo is touching the mouth with his right hands.
Frame 5: Pororo seems depressed with his bulging mouth. Pororo is muttering with unsatisfied gesture.

Frame 1: Eddy orders the robot to stop singing and dancing. the robot stops singing and dancing.
Frame 2: Eddy is so happy and proud that Eddy made the robot.
Frame 3: Eddy is imagining how surprised all his friends would be to see the robot Eddy made. Eddy seems really excited.
Frame 4: Eddy runs out to show his robot to his friends.
Frame 5: Pororo and Crong is calling Eddy and getting into Eddy's house.

Figure 9.8.2: More Story Generation Examples using our model *MaskGST-CG$_{\pm}$ /w aug. captions.*

Chapter 10

Conclusion and Future Directions

10.1 Conclusion

In this thesis we investigated the use of MaskGIT-style Transformers for the Task of Story Visualization, for the first time. The SOTA results, we achieved in various metrics provide evidence for the merit of our approach in the task of SV. Additionally, they indicate that such architectures should be further investigated for generative vision tasks, in general.

Specifically, we devised our baseline MaskGST model, based on MaskGIT, with additional cross-attention mechanisms, to allow the generated images at each stage in the story, to be influenced by past and future images. We experimented with various modifications of the initial architecture.

Several of those experiments failed to yield improvements compared to the baseline. Using T5-XXL as a text encoder is probably not optimal because of the niche text descriptions of our dataset, that are heavily dominated by unusual character names. We suspect that our attempt to perform Latent Super Resolution failed, because the higher resolution image tokens are too detailed for the Transformer to predict based on simple text prompts. Regarding the Character-Attentive Token-Critic that we devised, we believe that it fell short because conditioning solely on character embeddings is not expressive enough. Finally, we suspect that our attempt to perform latent space disentanglement of character features, from background features failed because it ended up being too confusing for the Transformer to model.

On the other hand, some of our experiments gave promising results. Integrating SV-layers into the Transformer benefited all metrics, by treating all the image and text tokens of a story as a continuous sequence, for a portion of the generative process. Secondly, we proposed a simple technique for image-agnostic caption augmentation, using an LLM. This idea improved our results by reducing the risk for over-fitting and helping the model focus on important textual concepts. Finally, the Character Guidance method we devised showed the most promising results. By forming three separate sets of logits based on text-conditions, the positive Character set and the negative Character set and then combining them, it succeeds in specifically guiding the model towards high quality character generation, whilst maintaining other information from the captions.

By combining promising methods and tuning hyper-parameters we arrived at our best model, *MaskGST-CG$_{\pm}$ w/ aug. captions (d=2048)*. This model, ourperforms the previous SOTA transformer by 9.3, 11.8 and 12.8 points in terms of FID, Char-F1 and Char-Acc, respectively. Despite being larger than previous Transformers it is more time-efficient both at training and inference.

10.2 Future Directions

We believe that our results strongly indicate the competence of MaskGIT-style transformers for the Task of Story Visualization. Our work paves the way for further experimentation with such architectures, either by scaling up the size of our Models, or by exploring other possible modifications. Besides, MaskGIT

architectures have seen relatively scarcely studied even in Text-to-Image generation settings. We hope that our work can encourage their exploration, in that realm as well.

Moreover, our caption augmentation method, provides an easy-to-use setting for enriching datasets that include text, including, but not limited to Story Visualization and Text-to-Image Generation. Besides, as Large Language Models become more available, our method could be extended by extracting alternative captions, using multiple LLMs, to achieve higher heterogeneity in terms of text descriptions.

Regarding our Character Guidance method, we maintain that it deserves further investigation. On the one hand, it can be tested in different settings, for the task of Story Visualization. One possible way of doing this would be to integrate it into (large) pre-trained models, either diffusion or Transformer based. This would include adding a positive and a negative embedding for each character, to the model and fine-tuning it, using our training scheme, where the text conditions are discarded for a portion of the training samples. Since the number of additional parameters is small, this could work with relatively little extra training. We presume that combining better prompt understanding, due to long pre-training, with our effective character guidance method could yield impressive results for the task.

Finally, another possible research path would be the generalization of our Character Guidance method to other tasks. Specifically, any generative task where we are explicitly interested in the generation of a specific set of concepts (Characters in the case of SV) could possibly be benefited by the adoption of such a method.

Chapter 11

Bibliography

[1] Ahn, D. et al. "Story visualization by online text augmentation with context memory". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 3125–3135.

[2] Bauer, L., Wang, Y., and Bansal, M. *Commonsense for Generative Multi-Hop Question Answering Tasks*. 2019. arXiv: 1809.06309 [cs.CL].

[3] Chang, H. et al. "Maskgit: Masked generative image transformer". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11315–11325.

[4] Chang, H. et al. "Muse: Text-to-image generation via masked generative transformers". In: *arXiv preprint arXiv:2301.00704* (2023).

[5] Chen, H. et al. "Character-centric Story Visualization via Visual Planning and Token Alignment". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 2022, pp. 8259–8272.

[6] Dai, H. et al. *AugGPT: Leveraging ChatGPT for Text Data Augmentation*. 2023. arXiv: 2302.13007 [cs.CL].

[7] Ding, M. et al. "Cogview: Mastering text-to-image generation via transformers". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 19822–19835.

[8] Esser, P., Rombach, R., and Ommer, B. "Taming transformers for high-resolution image synthesis". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 12873–12883.

[9] Fan, L. et al. "Improving clip training with language rewrites". In: *Advances in Neural Information Processing Systems* 36 (2024).

[10] Feng, Z. et al. "Improved Visual Story Generation with Adaptive Context Modeling". In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 4939–4955. DOI: 10.18653/v1/2023.findings-acl.305. URL:

[11] Goodfellow, I. et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[12] Ho, J. and Salimans, T. "Classifier-Free Diffusion Guidance". In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. 2021.

[13] Jiang, A. Q. et al. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL].

[14] Kim, K.-M. et al. *DeepStory: Video Story QA by Deep Embedded Memory Networks*. 2017. arXiv: 1707.00836 [cs.CV].

[15] Lei, J. et al. "MART: Memory-Augmented Recurrent Transformer for Coherent Video Paragraph Captioning". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 2603–2614.

[16] Lezama, J. et al. "Improved masked image generation with token-critic". In: *European Conference on Computer Vision*. Springer. 2022, pp. 70–86.

[17] Li, J. et al. "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation". In: *International conference on machine learning*. PMLR. 2022, pp. 12888–12900.

[18] Li, Y. et al. "Storygan: A sequential conditional gan for story visualization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 6329–6338.

[19] Maharana, A. and Bansal, M. "Integrating Visuospatial, Linguistic, and Commonsense Structure into Story Visualization". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 6772–6786.

[20] Maharana, A., Hannan, D., and Bansal, M. "Improving Generation and Evaluation of Visual Stories via Semantic Consistency". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2021, pp. 2427–2442.

[21] Maharana, A., Hannan, D., and Bansal, M. *StoryDALL-E: Adapting Pretrained Text-to-Image Transformers for Story Continuation*. 2022. arXiv: 2209.06192 [cs.CV].

[22] Pan, X. et al. "Synthesizing coherent story with auto-regressive latent diffusion models". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2024, pp. 2920–2930.

[23] Radford, A. et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[24] Raffel, C. et al. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[25] Rahman, T. et al. "Make-a-story: Visual memory conditioned consistent story generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 2493–2502.

[26] Ramesh, A. et al. "Zero-shot text-to-image generation". In: *International conference on machine learning*. Pmlr. 2021, pp. 8821–8831.

[27] Ramesh, A. et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV].

[28] Rombach, R. et al. "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[29] Ronneberger, O., Fischer, P., and Brox, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[30] Saharia, C. et al. "Photorealistic text-to-image diffusion models with deep language understanding". In: *Advances in neural information processing systems* 35 (2022), pp. 36479–36494.

[31] Selvaraju, R. R. et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. DOI: 10.1007/s11263-019-01228-7. URL:

[32] Sennrich, R., Haddow, B., and Birch, A. "Neural machine translation of rare words with subword units". In: *arXiv preprint arXiv:1508.07909* (2015).

[33] Shen, X. and Elhoseiny, M. *StoryGPT-V: Large Language Models as Consistent Story Visualizers*. 2023. arXiv: 2312.02252 [cs.CV].

[34] Sohl-Dickstein, J. et al. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International conference on machine learning*. PMLR. 2015, pp. 2256–2265.

[35] Song, T. et al. *Causal-Story: Local Causal Attention Utilizing Parameter-Efficient Tuning For Visual Story Synthesis*. 2023. arXiv: 2309.09553 [cs.CV].

[36] Song, Y.-Z. et al. "Character-Preserving Coherent Story Visualization". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020.

[37] Touvron, H. et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[38] Ubani, S., Polat, S. O., and Nielsen, R. *ZeroShotDataAug: Generating and Augmenting Training Data with ChatGPT*. 2023. arXiv: 2304.14334 [cs.AI].

[39] Van Den Oord, A., Vinyals, O., et al. "Neural discrete representation learning". In: *Advances in neural information processing systems* 30 (2017).

[40] Van den Oord, A. et al. "Conditional image generation with pixelcnn decoders". In: *Advances in neural information processing systems* 29 (2016).

[41] Van Den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. "Pixel recurrent neural networks". In: *International conference on machine learning*. PMLR. 2016, pp. 1747–1756.

[42] Vaswani, A. et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[43] Whitehouse, C., Choudhury, M., and Aji, A. F. *LLM-powered Data Augmentation for Enhanced Cross-lingual Performance*. 2023. arXiv: 2305.14288 [cs.CL].

[44] Wozniak, S. and Kocon, J. "From Big to Small Without Losing It All: Text Augmentation with ChatGPT for Efficient Sentiment Analysis". In: *2023 IEEE International Conference on Data Mining Workshops (ICDMW)*. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2023, pp. 799–808. DOI: 10.1109/ICDMW60847.2023.00108. URL:

[45] Xu, T. et al. "Attngan: Fine-grained text to image generation with attentional generative adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1316–1324.

[46] Zhang, H. et al. "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5907–5915.

[47] Zhang, R. et al. "The unreasonable effectiveness of deep features as a perceptual metric". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.