Vibe Coding: Formation Starter (1j)

v1 Philippe Pary - Thomas Foutrein 2025

Première approche d'une formation Vibe Coding en une journée Formation "Vibe Coding" pour Développeurs Débutants

Une journée pour apprendre à coder avec l'assistance de l'IA

Pré-requis

- Connaissance d'au moins un langage de programmation ou scripting
- Connaissance des processus de développement informatique
- Connaissance du contrôle de version (git)
- Connaissance des IDE (VSCode, IntelliJ)
- Matériel de travail adapté à la programmation
- VScode installé
- Cursor installé
- Licence Copilot./ Cursor (fournies pour la durée de la formation)

Objectifs de la formation

- Coder avec Cursor
- Améliorer du code existant avec Cursor
- Intégrer Cursor dans un environnement de développement existant
- Personnaliser Cursor à vos besoins métier
- Formuler des prompts efficaces pour l'IA dans un contexte de développement
- Développer un projet complet en utilisant les techniques de "Vibe Coding"

Cours

9h00 - 9h30 : Introduction et présentation

Tour de table:

- Nom, fonction
- As-tu déjà vibe codé ?
- Comment vois-tu le vibe coding ?
- Ta machine est prête ? (Cursor installé) / accès machine AWS / accès au CodeLab
- Quelles sont tes attentes vis à vis de la journée ?

➡️ Présentation du plan de la journée

9h30 - 10h15 : Information : Les outils du "Vibe Coding" (Théorie)

Frimer

Démonstration en direct d'une création rapide d'application en utilisant le vibe coding. Nous allons générer un environnement containerisé pour notre codelab directement avec l'IA. Nous examinerons également des exemples concrets de projets développés lors des sessions précédentes pour illustrer l'efficacité de cette approche.

- vibe-coder le codelab de la journée (générer un container)
- présenter les sites des sessions précédentes

Comment fonctionne l'IA pour le code et le vibe coding

Concept récent, de février 2025. Par Andrej Karpathy (co-fondateur d'OpenAI)

Technique de programmation assisté par des LLMs et basé sur du prompt. Il se différencie d'un outil de complétion basé sur les LLMs comme Copilot par l'importance crucciale du prompt dans le processus de développement.

Le vibe-coding peut être utilisé comme un outil low/no-code. Ceci dit, la pratique des derniers mois démontre que comme tous les outils low/no-code, un non-informaticien va produire des logiciels de mauvaise qualité de cette façon (failles de sécurités, bugs…). Cette pratique est à réserver pour les POC ou le prototypage.

Entre les mains d'un développeur compétent, les outils de vibe-coding entraînent une hausse de la productivité et de l'efficacité. Mais également une transformation du métier.

On voit apparaître une forme de fatigue mentale. Les choses vont vite, le cerveau doit encaisser des changements rapides. Des opportunités se présentent dans tous les sens. Quelques conseils: - plus que jamais, donner les priorités sur le backlog (PM) et les respecter (développeurs) - savoir s'arrêter et prendre du recul - discuter avec les collègues sur les évolutions récentes - se rappeler du besoin de relire le code généré, ne fut-ce que pour en avoir une connaissance même superficielle (et donc pouvoir le maintenir)

Il existe aussi de l'auto-complétion boostée à l'IA. Comme Copilot dans sa version historique. Ça n'est pas du vibe-coding, juste une auto-complétion très riche et efficace qui permet d'éviter d'avoir à écrire du code redondant. Le vibe coding se définit par l'usage du prompt.

Les LLM : comprendre les fondements

Les Large Language Models (LLM) sont des modèles d'intelligence artificielle entraînés sur d'immenses corpus de textes et de code pour prédire la suite la plus probable d'une séquence donnée. Contrairement aux systèmes experts traditionnels qui fonctionnent

avec des règles prédéfinies, les LLM apprennent des motifs statistiques à partir des données.

Au cœur de leur architecture se trouvent des réseaux de neurones profonds basés sur l'attention (transformers), capables de capturer des dépendances à longue distance dans le texte ou le code. Ces modèles encodent l'information en "tokens" - des unités qui peuvent représenter des mots, parties de mots, ou symboles spéciaux. Un prompt de programmation typique peut contenir plusieurs centaines ou milliers de tokens.

L'apprentissage d'un LLM se déroule en plusieurs phases:

1. **Pré-entraînement**: le modèle est exposé à d'énormes quantités de texte et code provenant d'internet, dépôts GitHub, documentation technique, etc.
2. **Fine-tuning**: le modèle est affiné sur des tâches spécifiques, comme la génération de code
3. **RLHF (Reinforcement Learning from Human Feedback)**: les réponses du modèle sont évaluées par des humains, créant un signal d'apprentissage pour améliorer la qualité

Pour la génération de code, ces modèles ont été spécifiquement entraînés sur des millions de repositories publics, questions StackOverflow, et documentation technique. Ils ont ainsi "mémorisé" des patterns de programmation, des structures algorithmiques, et des solutions communes à des problèmes récurrents.

Le processus de génération de code

Lorsque vous soumettez un prompt à un LLM dans un contexte de vibe coding, le processus suivant se produit:

1. **Tokenisation**: votre prompt est découpé en tokens compréhensibles par le modèle
2. **Contextualisation**: le modèle analyse le prompt dans son contexte (code existant, fichiers ouverts)
3. **Prédiction auto-régressive**: le modèle génère un token après l'autre, chaque nouveau token étant influencé par tous les précédents
4. **Échantillonnage**: plutôt que de toujours choisir le token le plus probable, le modèle introduit une certaine randomisation (via la température) pour favoriser la créativité

Ce processus est fondamentalement probabiliste - le modèle propose ce qu'il "pense" être la suite la plus plausible, mais il n'a pas de compréhension profonde du code qu'il génère. C'est pourquoi les LLM peuvent produire du code qui semble correct mais contient des erreurs subtiles ou des failles de sécurité.

Les limites intrinsèques des LLM pour le code incluent: - **"hallucination"** de fonctionnalités ou API inexistantes - L'incapacité à exécuter ou tester directement le code qu'ils génèrent - Une connaissance limitée aux données de leur entraînement (cutoff knowledge) - Des difficultés avec des problèmes très spécifiques ou nouvellement émergents

Ces limitations soulignent l'importance du développeur humain dans le processus de vibe coding, qui doit rester vigilant et critique face aux propositions de l'IA.

Fonctionnalités du vibe-coding

- Autocomplétion sous stéroïdes
- Mode ask et variantes (*architect* chez Kilo Code qui va conceptualiser tout un logiciel à partir d'une demande métier)
- Mode prompt et variantes (*edit* chez Copilot qui agit *bêtement* se contentant d'appliquer strictement les instructions sur une zone de code donnée)
- tools : compléter les limites des LLM (création d'image, recherche sur le web etc.)

ce qu'on peut demander

Ask (Consulter sur…)	**Prompt (Générer…)**
Les meilleurs choix technologiques	Du code
Les solutions algorithmiques	Des tests
Les solutions architecturales	Relire / auditer du code
Les solutions de design	Des commentaires
Les solutions de sécurité	Des documentations
Les solutions de performance	Des diagrammes
Les solutions de scalabilité	… vous avez l'idée à force
… vous avez l'idée à force	
La meilleure manière de prompter	

Workflow du développeur avec de l'IA

Le workflow traditionnel de développement se transforme radicalement avec le vibe coding. Au lieu de passer des heures à écrire du code ligne par ligne, le développeur devient un architecte et un guide pour l'IA.

Le processus typique inclut:

1. Définition précise des besoins et objectifs
2. Rédaction d'un prompt détaillé et structure

3. Dialogue itératif avec l'IA pour affiner le code généré
4. Validation et tests à chaque étape significative
5. Refactoring et optimisation en collaboration avec l'IA

Dans tous les cas, l'écrasante majorité du code sera rédigée par LLM. Quelques bugs ou fonctionnalité peuvent être traîtées à la main, soit que le LLM s'y casse les dents, soit qu'il soit plus rapide, plus fun ou instructif de faire ainsi.

Soyez intentif aux checkpoints et le besoin de revenir en arrière par moments: Établissez des points de validation réguliers dans votre processus de développement. N'hésitez pas à créer des branches Git à chaque étape fonctionnelle complétée. Si l'IA commence à produire des résultats insatisfaisants, vous pourrez facilement revenir à une version stable et reprendre depuis un point connu.

Le métier de développeur consiste surtout à s'assurer de la qualité du code produit. La partie relecture de code, comme pour une PR/MR, devient centrale.

1. Itératif

Étape par étape. On commence par un gros prompt (spec, cahier des charges etc) qui défini le projet. On ajoute les fonctionnalités (ou les refacto) un à un à partir de prompts successif.

✅ Meilleure maîtrise de l'évolution du logiciel
❌ Pensez à commiter chaque étape fonctionnelle, le vibe-coding peut vite partir en vrille

2. One big happy prompt

On prépare un prompt très détaillé, généralement sous la forme d'un fichier markdown et on accompagne l'IA dans la réalisation du projet

✅ Peu de risques de régressions entre les prompts: le LLM sait où il va. Le projet a été pensé et structuré
❌ La relecture et bonne compréhension de ce qui a été généré est longue et fastidieuse. Le temps de développement est généralement augmenté

Les IDE disponibles

Outil	Type	Avantages	Inconvénients	Modèles	URL d'accès
Cursor	Éditeur dédié	✅ Interface optimisée pour le vibe coding ✅ Règles personnalisables	⚠️ Application séparée à installer	Claude, GPT-4, autres	cursor.com

Outil	Type	Avantages	Inconvénients	Modèles	URL d'accès
		✅ Choisi par Astek			
gemini-cli	Outil CLI	✅ Scriptable, idéal pour l'automatisation	⚠️ Pas d'interface graphique	Gemini	github.com/google/gemini-cli
		✅ Gratuit			
Replit	Éditeur en ligne	✅ Environnement complet avec déploiement	⚠️ Nécessite une connexion internet	Plusieurs modèles propriétaires	replit.com
		✅ Collaboration en temps réel	⚠️ Limitations dans version gratuite		
WindSurf	Éditeur dédié	✅ Optimisé pour la visualisation	⚠️ Relativement récent	Gemini, Claude	windsurf.com
		✅ Interface moderne	⚠️ Moins de documentation		
Claude Code	Interface web	✅ Puissance du modèle Claude	⚠️ Pas d'intégration IDE	Claude uniquement	anthropic.com/claude-code
		✅ Interface dédiée au code	⚠️ Nécessite un abonnement		
Kilo Code	Extension VSCode	✅ Modes dédiés pour le code	⚠️ Configuration requise	GPT-4 et autres	marketplace.visualstudio.com/kilocode
		✅ Léger	⚠️ Fonctionnalités plus limitées		
Continue	Extension VSCode	✅ Bonne intégration workflow	⚠️ Performance variable	Plusieurs modèles	marketplace.visualstudio.com/continue
		✅ Open source	⚠️ Moins stable		

Outil	Type	Avantages	Inconvénients	Modèles	URL d'accès
Copilot	Extension multi-IDE	✅ Largement adopté, mature ✅ Bonne intégration	⚠️ Coût de licence ⚠️ Moins orienté vibe coding	GPT-4 (principalement)	github.com/features/copilot

Comment fonctionne l'IA pour le code

Notions essentielles à comprendre:

- LLM (Large Language Models): modèles fondations qui ont été pré-entraînés sur d'immenses corpus de données textuelles, puis affinés pour comprendre et générer du code
- Data source: les sources utilisées pour l'entraînement (GitHub, StackOverflow, documentation technique)
- Vectorisation: conversion du texte en représentations numériques que l'IA peut traiter
- Prompt engineering: l'art de formuler des instructions qui maximisent la qualité des réponses de l'IA
- MCP (Multi-modal Code Processing): capacité à traiter simultanément le code et d'autres formes de contenu comme les images ou diagrammes
- Agents IA: systèmes qui peuvent planifier, exécuter et réfléchir sur des tâches complexes de programmation de manière autonome.

(- Agent2Agent, abordé dans les formations avancées)

Comme pour tout sujet, les LLM fonctionnent en perroquets stochastiques : ils analysent un texte donné, réduit en tokens, et tentent de déduire l'intention entre le prompt (dont prompts système et rules) et la codebase.

Le LLM tente alors de générer la réponse la plus probable, sur base de son entraînement et de son nombre de paramètres possibles._

Aspect	Vibe Coding	Coding Traditionnel
Approche	Décrire les résultats souhaités en langage courant	Écrire du code ligne par ligne avec des langages de programmation
Outils Utilisés	Grands Modèles de Langage (LLM)	IDE, compilateurs, débogueurs et frameworks de codage
Niveau de	Adapté aux débutants et non-	Nécessite de solides connaissances

Aspect	Vibe Coding	Coding Traditionnel
Compétence	programmeurs	techniques
Connaissance en Programmation	Minimale - se concentre sur l'intention, pas la syntaxe	Élevée - l'utilisateur doit comprendre la logique, la syntaxe et la structure
Vitesse	Rapide - idéal pour constructions et itérations rapides	Plus lent - implique plus de précision et de planification
Contrôle et Précision	Inférieur - dépend de l'interprétation de l'IA	Élevé - chaque ligne est écrite et revue par le développeur
Idéal Pour	Prototypage rapide, petits outils, exploration d'idées	Systèmes évolutifs, sécurisés et de qualité production
Processus de Révision	Sortie revue après génération; souvent itératif	Le code est revu pendant et après le développement
Sécurité & Maintenance	Risque plus élevé dû aux inconnues dans le code généré	Plus facile à sécuriser et maintenir avec une compréhension complète
Courbe d'Apprentissage	Faible - les utilisateurs peuvent commencer à construire avec des conseils de base	Élevée - nécessite d'apprendre langages, logique et débogage

Modèles à favoriser

- **Claude 4 Sonnet** (Opus est plus cher, limité, mais bien meilleur)
- Gemini-2.5-pro donne de bons résultats
- ChatGPT-4.1 sur le front

C'est aussi très subjectif, d'un dev à l'autre on apprécie des choses différentes. Un LLM plus ou moins verbeux, réflexif…

➡️ Faire un essai en one try des différents modèles avec le prompt `Développe un système de gestion de liste de tâches à partir de fichiers markdown en utilisant Python3, HTML, Javascript (sans framework) et TailWind`

Git et Vibe coding

%%%TBD%%%

Principales Limites et pièges à éviter

- Comitez ! L'IA peut ruiner en quelques minutes plusieurs semaines de travail
- Faites des branches locales en grande quantité pour tester plusieurs prompts et modèles pour régler un même point
- Soyez attentif à la couverture des tests - surtout sur les modèles plus faibles

- Soyez attentif à la relecture de code - n'hésitez pas à faire relire le code généré par plusieurs modèle mais surtout via une nouvelle session de chat (utilisez le mode Ask AI)
- Soyez attentif à la sécurité - les modèles peuvent générer du code défectueux ou malveillant
- Privilégiez les LLMs connus, renseignez vous sur les LLMs plus secondaires ou entraînés par votre client (ou pire, des inconnus)
- Vous pouvez interrompre le processus de génération de code à tout moment si vous considérez que le LLM part dans la mauvaise direction

10h30 - 11h00 : Information : L'art du prompt engineering pour le code (Théorie)

Principes fondamentaux pour communiquer avec l'IA

On tourne autour du prompt. Le prompt a plusieurs niveaux

Chaque modèle d'IA possède des instructions par défaut qui définissent son comportement général. Ces instructions système sont invisibles pour l'utilisateur mais influencent fortement les réponses. Les outils comme Cursor permettent de personnaliser ces instructions à différents niveaux pour obtenir des résultats plus adaptés à vos besoins spécifiques.

- Rules générales: Users rules dans la configuration. S'appliqueront à tous les projets. Vous pouvez par exemple lui demander d'être didactique ou à l'inverse laconique. Peuvent être générales à l'entreprise.
- Rules locales: dans un dossier `.cursor/rules` s'applique au dossier et aux sous-dossiers. Utile en cas de mono-repo pour appliquer un comportement différent suivant le dossier (language de programmation, règles de développement dont convention de nommage des variables, contexte général du projet). Peuvent être générales à un projet.
- Utiliser `context7` pour avoir de meilleurs résultats https://github.com/upstash/context7
- Le prompt en tant que tel

Pour les rules consultez:

Application	Documentation des règles / configuration du prompt
:———	:————————————————
Cursor	Rules & Context
Replit	Replit AI Documentation
WindSurf	WindSurf Documentation
Claude Code	Claude Code Prompt Guide
Kilo Code	Kilo Code Configuration

| **Continue** | Continue Rules & Context |
| **Copilot** | GitHub Copilot Documentation |

Structure d'un prompt efficace pour le développement

Très proche d'une formalisation des conventions de développement d'un projet : DOR/DOD, project manifesto…

Favoriser le mono-repo:

- Avantages: meilleure compréhension du contexte global par l'IA, cohérence entre les différentes parties du projet, facilité pour référencer d'autres composants.
- Inconvénients: perte du contexte historique de chat lors des changements de dossier, besoin de partager la documentation entre les projets.

Solutions: utiliser des serveurs MCP avec accès Git, intégrer des liens vers la documentation Swagger, maintenir des fichiers texte de référence accessibles à l'IA, et configurer des règles spécifiques par dossier.

Rappeler - contexte général du projet : contexte général du projet, objectifs, contraintes, etc. fonctionnel et technique - exemples de données (et contre-exemples) - exemples de règles métier (et contre-exemples) - technologies utilisées - dans quel dossier travailler (et préciser si l'IA doit s'y contraindre, elle a tendance à aller un peu partout sinon)

Exemples de prompts qui fonctionnent vs ceux qui échouent

Évitez les prompts vagues et non techniques comme "Fais-moi un site web pour vendre des gâteaux". Préférez des prompts structurés et précis comme "Développe une application e-commerce en React avec un backend Node.js qui permet de présenter un catalogue de pâtisseries, gérer un panier d'achat, et traiter des paiements via Stripe. Voici la structure de données pour les produits: […]".

Adapter ses prompts selon le contexte et l'objectif

Avant de vous lancer dans un développement complexe, utilisez le mode "Ask" pour interroger l'IA sur la meilleure façon de structurer votre prompt principal. Par exemple: "Je souhaite développer une application de gestion de tâches. Quelles informations devrais-je inclure dans mon prompt pour obtenir le meilleur résultat possible?"

Exemple task list avec des objectifs plus ou moins grands: - Pour un projet personnel simple: Prompt concis avec fonctionnalités de base - Pour une application d'entreprise: Prompt détaillé incluant architecture, sécurité, et conformité - Pour un déploiement AWS: Ajout de sections spécifiques sur l'infrastructure cloud et les services AWS à utiliser - Pour

un projet en équipe: Inclusion des conventions de code, structure des branches Git, et processus de revue

11h00 - 12h30 : Compréhension : Premiers pas pratiques (Pratique)

Exercices simples de génération de code

- Générer un snake en HTML/CSS/JS (sans Framework) avec sauvegarde des meilleurs score en `localStorage`
- Calculer en python une approximation de Pi en utilisant la méthode de Monte-Carlo
- Création d'une application client/serveur de gestion des tâches
- Bot Discord qui indique la météo à partir d'une commande `/meteo <ville>`

Mini-défis pour tester différentes approches de prompts

avoir des repos buggy en PHP, Java et React %%%faire générer les projets, les saborder à la mano%%%

avoir une suite de Fibonacci en boucle infinie à partir d'un prompt sain

13H30-14H00 : Compréhension : retour expérience (Théorie)

Tour de table: * Qu'as-tu appris ce matin ? * Quelles difficultés as-tu rencontrées ? * Qu'est-ce qui t'a le plus marqué ?

14h00 - 14h30 : Compréhension: Planification de projet (Théorie)

- Tour de table:
  - Choisir un projet

14h30 - 16h00 : Utilisation : Développement assisté par IA (Pratique)

- Consulter l'IA en mode Ask pour rédiger un plan de développement
- Développer le projet
- Affiner son guidage l'IA
- Débugguer avec l'IA (cmd+L ou ctrl+L pour envoyer les logs au prompt)
- Générer la documentation

16h15 - 17h15 : Maîtrise : Finalisation et optimisation (Pratique)

- Revue de code assistée par IA: Passez en mode Ask, rédiger un prompt `Tu es un développeur expert, tu es mon binôme et tu relis mon code. Liste ici toutes les améliorations fonctionnelles, dans l'usage de librairies, dans le code que tu pourras trouver. Pointe les bonnes pratiques à adopter. Souligne les problèmes de sécurité que tu as repérés`
- Optimisation des performances: consulter l'IA (mode Ask) et appliquer les recommandations

- Tests automatisés générés par l'IA: consulter l'IA (mode Ask) et faites lui rédiger les tests. Passer en revue, compléter (ou faire compléter)
- Finalisation du projet: faire rédiger, ou relire, la documentation.
- Analyse critique du processus: demandez à l'IA d'analyser le processus de développement suivi et de proposer des améliorations pour un projet futur
- Documentation des apprentissages: créez un document résumant les techniques de prompt engineering qui ont fonctionné et celles à éviter

17H15-17H30 : Risques à connaître (Théorie) - facultatif

Développement durable et Green IT avec le Vibe Coding

Le vibe coding a des implications importantes sur l'empreinte carbone du développement logiciel. D'un côté, l'utilisation intensive des LLM augmente la consommation énergétique liée à l'inférence sur des serveurs distants. De l'autre, cette approche peut conduire à des applications plus optimisées et à un cycle de développement raccourci.

Pour un vibe coding respectueux de l'environnement : - Optimisez vos prompts pour réduire le nombre d'interactions nécessaires - Privilégiez les modèles plus légers lorsque la complexité de la tâche le permet - Demandez explicitement à l'IA de générer du code efficient en ressources - Incluez des critères d'éco-conception dans vos prompts (optimisation des requêtes, minimisation des dépendances, etc.) - Validez la performance énergétique du code généré avec des outils dédiés

Un développeur responsable doit être conscient que chaque prompt a un coût énergétique. Une étude récente estime qu'une session de vibe coding intensive peut consommer l'équivalent de plusieurs heures d'utilisation d'un ordinateur portable. Intégrez cette dimension dans vos choix quotidiens en privilégiant des approches qui maximisent l'efficacité tout en minimisant l'empreinte environnementale.

L'IA peut également être un allié pour optimiser l'efficience énergétique de vos applications. Demandez-lui spécifiquement d'analyser et d'améliorer votre code sous cet angle, et vous obtiendrez souvent des suggestions pertinentes pour réduire la consommation de ressources.

Sécurité : les défis du code généré par IA

Le vibe coding introduit de nouveaux défis de sécurité que tout développeur doit connaître. Les LLM peuvent involontairement générer du code vulnérable ou même malveillant, particulièrement lorsqu'ils sont mal dirigés ou manipulés.

Le MCP poisoning (Model Context Poisoning) est une préoccupation majeure : des acteurs malintentionnés peuvent injecter des instructions cachées dans les prompts ou

dans le code source pour manipuler les modèles. Pour vous protéger : - Vérifiez systématiquement le code généré avant de l'exécuter - Utilisez des outils d'analyse statique de code sur les outputs de l'IA - Ne partagez jamais de données sensibles ou de secrets dans vos prompts - Restez vigilant face aux comportements inhabituels de l'IA - Appliquez le principe de moindre privilège pour toute exécution de code généré

La confiance accordée au LLM doit toujours être mesurée. Si les grands modèles comme Claude ou GPT sont généralement fiables, ils ne sont pas infaillibles. Les risques de fuite de données existent également : certains modèles peuvent mémoriser des informations sensibles de vos prompts et les réutiliser ailleurs. Utilisez des instances privées ou des solutions on-premise pour les projets critiques.

Intégrez systématiquement une revue de sécurité dans votre workflow de vibe coding, idéalement en demandant à l'IA elle-même d'analyser les vulnérabilités potentielles du code qu'elle a généré.

Considérations éthiques du Vibe Coding

L'utilisation du vibe coding soulève des questions éthiques importantes que chaque développeur doit considérer. Les modèles d'IA reflètent les biais présents dans leurs données d'entraînement, ce qui peut conduire à des solutions techniques qui perpétuent des inégalités existantes ou qui manquent d'inclusivité.

Le sexisme, le racisme, le validisme, l'eurocentrisme et l'occidentalocentrisme sont particulièrement prégnants dans de nombreux LLM. Le code généré peut privilégier des approches, des normes ou des hypothèses qui ne sont pas universelles.

Par exemple: - la gestion des dates, des noms ou des adresses peut ne pas tenir compte des spécificités culturelles diverses (i18n claquée au sol) - d'éventuels emoji peuvent être bloqués en couleur de peau et en genre (pour rappel, la bonne pratique étant de choisir des emoji neutres) - des formulations inclusives : « madame / monsieur » d'un courier changé en juste « monsieur », « H/F » d'une offre d'emploi qui disparaît - l'IA peut générer des applications qui ne respectent pas le RIAA (alt des images pourris, texte essentiel accessible uniquement avec un effet visuel lié à JavaScript etc.)

Pour pratiquer un vibe coding éthique : - Examinez le code généré pour identifier et corriger les biais potentiels - Diversifiez vos exemples et cas d'usage dans vos prompts - Spécifiez explicitement vos attentes en matière d'inclusivité et d'accessibilité - Testez les solutions avec des perspectives et des données diverses - Restez critique face aux solutions "standard" proposées par l'IA

La transparence est également un enjeu majeur : dans la documentation, signalez clairement aux utilisateurs finaux quelles parties du code ont été générées par IA et

comment elles ont été validées. Cette pratique renforce la confiance et permet une utilisation plus responsable de la technologie.

Aspects juridiques et réglementaires

Le vibe coding s'inscrit dans un cadre juridique en pleine évolution qu'il est essentiel de maîtriser. RGPD, droit d'auteur, licences libres etc. il y a de nombreux aspects juridiques pour l'instant mal pris en compte par les LLM.

Pour vous protéger légalement : - Documentez votre processus de vibe coding (gardez l'historique des prompts et les modèles utilisés) - Vérifiez les licences des modèles que vous utilisez - Soyez attentif aux licences mentionnées dans le code généré - Réécrivez ou adaptez le code lorsque nécessaire pour éviter les problèmes de droits - Consultez un expert juridique pour les projets d'envergure

La conformité avec les réglementations sectorielles (finance, santé, etc.) doit également être prise en compte dès la conception de vos prompts, ie rejoindre vos prompts système/projet.

17h30 - 18h00 : Bilan et perspectives (Théorie et discussion)

- Présentation des projets réalisés (tour de table)
  - Quelles difficultées as-tu rencontrées ?
  - Quelles bonnes pratiques as-tu retenues ?
  - As-tu des ressources à partager ?
  - As-tu des questions ?
- Évaluation de la formation (10 minutes)
- Rappel: le vibe est une méthode récente en constante évolution : maintenez une veille technologique sur le sujet